SQL Server 2000 Sizing and Capacity Planning Guidelines

Network Appliance, Inc. | Gerson Finlev | December 2004 | TR 3363

TECHNICAL REPORT

Network Appliance, a pioneer and industry leader in data storage technology, helps organizations understand and meet complex technical challenges with advanced storage solutions and global data management strategies.

Contents

1. Introduction

This technical report's primary focus is database sizing guidelines for deploying SQL Server 2000 databases utilizing NetApp storage. Enterprise-level SQL Server environments require careful planning driven by performance and reliability criteria that are the critical components for many SQL Server implementations. Servicing user transactions and delivering consistent and acceptable response times are critical for most applications utilizing SQL Server for storing and retrieving data from databases. Performance degradation can only be prevented when resource utilization is monitored and capacity planning guidelines are in place, well understood, and implemented properly. Therefore, capacity planning guidelines are an obvious part of this technical report.

Performance, sizing, and capacity planning guidelines are interrelated functional areas, with goals to drive the implementation of critical SQL Server database environments. It is important to configure a balanced system with no known performance bottlenecks even during peak loads. Most applications utilizing SQL Server for storing and retrieving information will not push the I/O throughput to the NetApp storage device's limit. However, if the storage device is a limiting performance factor, then all SQL Server operations will suffer. Having right-sized storage devices is essential for supporting high I/O throughput with low service time. It is the important baseline for developing capacity planning guidelines.

The focus of this technical report is storage requirements for SQL Server databases. Space requirements are fairly easy to calculate and not really an issue, but throughput and service time requirements are less exact and more difficult to define. The goal of this report is to educate and to create background knowledge that will make it easier to work with administrators to size storage correctly for both space and throughput. Correct sizing requires cooperation among all participants and is especially difficult to attain when consolidating both SQL Server instances and storage. Implementing new applications with SQL Server is especially challenging because I/O-related performance requirements are often not well understood. Performance and storage requirements depend on the workload generated by the SQL Server application, as well as on hardware configuration and memory utilization. It is very easy to illustrate how executing the exact same workload on the same Windows® server can result in very different physical I/O access patterns and thereby different storage requirements.

The three test results shown in Figure 1 and Figure 2 are driven by the exact same workload executing on the exact same system configuration. Test 1 in Figure 1 shows a read rate of around 8MB per second and a write rate of 12MB per second; the physical I/O load changes to a very different load and access pattern in test 3. It is even more fascinating that test 1 has almost saturated the filer (Figure 2) compared to test 3, in which the filer resources were not highly utilized. The Windows server's processor resources (four CPUs) were always 99% busy; these results will be discussed in detail in section 2. This report will discuss the throughput performance and explain the reason why it is not possible to provide exact information about the right storage-sizing guidance for throughput. This report will focus on the background knowledge that will make an implementation a performance success.

Figure 1 illustrates the results from executing the same application workload, resulting in very different physical I/O rate.

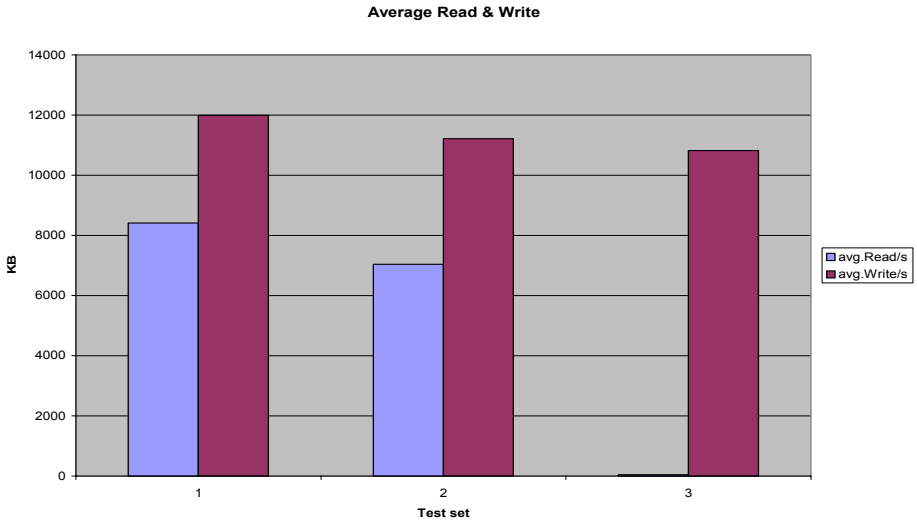

Figure 1) Average physical I/O.

Figure 2 illustrates the resulting filer utilization from executing the application workloads shown in Figure 1.

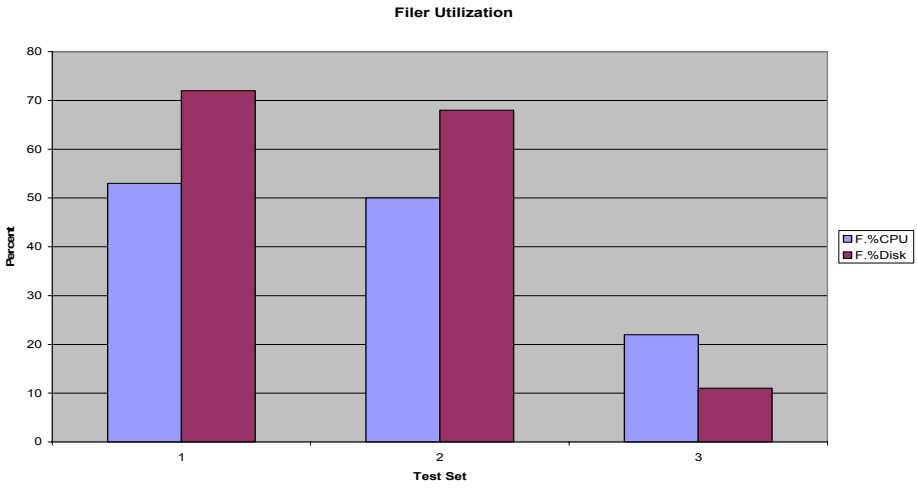

Figure 2) Average filer resource utilization.

The physical I/O rate appears to be related to the buffer cache hit ratio. If the hit ratio can be increased, then the physical I/O might decrease.

Developing sizing guidelines is a larger scale effort, and several factors have to be considered before arriving at a conclusion. This report allows for the fact that each customer case may be a unique scenario and need a customized analysis. However, considering a general approach, this report considers involving the following three related topics:

- Factors that affect SQL Server 2000 I/O access patterns
- Factors that effect NetApp storage device handling of I/O requests
- Correlating SQL Server and NetApp storage handling

This report discusses the sizing guidelines to explain the setup and configuration of the system before it is put into production mode, whereas capacity planning guidelines explain the procedure to maintain a healthy system over time by monitoring critical resources for optimum or peak performance. This report understands that performance doesn't explain what a good or poor performance measurement is; a performance counter is just a number. A counter that has an associated healthy value gives the term "performance counter" more meaning. The term "healthy counter" and its associated value are not objective criteria and will not be defined in this report, but it is something the administrator of a given environment has to think about. For example, is processor resource utilization of 100% a bad sign? The response may be that it may not be a bad sign in certain scenarios. If the system is running heavy batch loads during off hours, then it might not be a problem, but if the high utilization prevents important transactions from being serviced in a timely manner, there might be a performance problem that could result in upgrading the system or moving the execution of the batch jobs to another time of the day.

Developing capacity planning guidelines has to go through the following phases to achieve a better and optimal result:

- Collecting performance counters
- Defining actions on counter values
- Taking actions when a counter turns unhealthy

This guide is generic, with no specific application as a reference model; therefore, basic knowledge about I/O access methods used by SQL Server 2000 is important background information when right-sizing NetApp storage for specific applications.

Readers who are knowledgeable about SQL Server I/O handling may skip section 2, "Factors Affecting SQL Server's I/O Handling."

2. Factors Affecting SQL Server's I/O Handling

The primary purpose of a database is to store and retrieve data. Therefore, performing a lot of disk reads and writes is an inherent attribute of a database engine. Disk I/O operations consume many resources and take a relatively long time to complete. Much of the logic in relational database engines concerns making the pattern of I/O usage highly efficient. SQL Server uses a buffer cache to highly optimize disk access. The buffer cache is managed dynamically;[1] the size is adjusted according to available physical memory; other applications running on the server, including other SQL Server instances; and general transaction load. Database pages found in the buffer cache are logical I/O, and database pages that have to be retrieved from permanent storage are physical I/O.

2.1. Logical I/O (Buffer Cache) and the Effect on Physical I/O

SQL Server's storage engine will look for database pages in the buffer cache before reading from permanent storage. The more database pages found in the buffer cache, the more efficient SQL Server will be, and the higher the throughput SQL Server will be able to support.

It is easy to illustrate how the size of the buffer cache can drastically influence the transaction rate, which is affected by the changed logical I/O rate, which will indirectly change the physical I/O rate, access pattern, and storage device's utilization level. A simple test was executed using an eight-disk volume on an F960. The database contained 100 million rows, and the transaction

[1] It is possible to configure a fixed buffer cache size.

mix was 50% select and 50% update. A set of tests was executed with three different buffer cache sizes (50MB, 150MB, and 500MB). The only difference between the three tests was the size of the buffer cache. Changing the buffer cache size result in 40% higher transaction rate (Figure 3); the server's four processors were always 99% busy.

The main reason for the increased transaction rate is the changed cache hit ratio (rate of logical I/O) and access pattern and because the I/O subsystem had to work much harder with the small buffer cache. In fact, the changed buffer cache sizes completely changed the physical I/O pattern, from being about 45% read and 55% write to being almost 100% write (Figure 4).

Test Series	Buffer Cache	Transactions per Second
1	50MB	3045
2	150MB	3235
3	500MB	4250

Figure 3) Effect of buffer cache size (TPS).

When analyzing a given system's transaction rate, relate it to the buffer cache hit ratio. If the database is supporting an OLTP type of application, then the hit ratio ought to be greater than 80% The cache hit ratio with data warehouses might be even lower than 50%. If the hit ratio is too low, try to determine why. The larger the database is (relative to the buffer cache) and the more uniformly accessed the database is, the lower the buffer hit ratio will be; see section 2.1.1.

Figure 4 shows the filer's disk I/O (average and maximum sample during the 20-minute test runs). The eight-disk volume is fully utilized with the smaller buffer cache; it is highly utilized because of random read/write. The physical I/O changes to almost 100% sequential write with the larger buffer cache.

The difference between the average write rate and the maximum write rate is an indication of the effect of database checkpoints. A checkpoint will flush all modified database pages in SQL Server's buffer cache as quickly as possible. Of course, if the storage device is already working hard to keep up with the load, then the effect of the checkpoint will not be very noticeable, but otherwise a checkpoint is easily detected; test 3's average write rate was around 11MB per second, which is lower than the result of test set 1. There are two reasons for this difference: first, the value of a 10-second sample depends on the correlation between the checkpoint executed by the database engine and the filer's checkpoint; second, a bigger buffer cache will contain more modified pages, and each page might be modified multiple times between checkpoints.

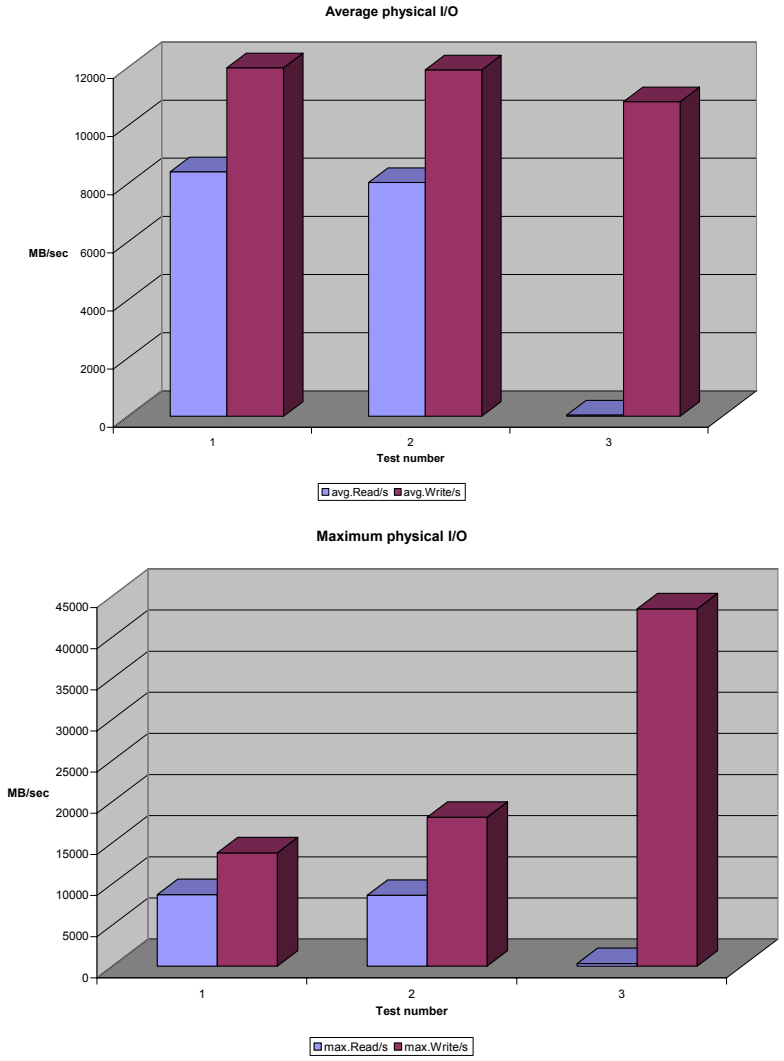

Figure 4) Effect of buffer cache size (MB).

The smaller buffer cache created a physical I/O load that highly utilized the disk drives and practically saturated the used disk subsystem (volume). The filer's CPU was also highly influenced by the size of the SQL Server buffer cache. The smaller buffer cache created a load on the filer that not only was very disk intensive but also highly utilized the filer's CPU (Figure 5).

The same effect that was illustrated by changing the buffer cache could have been illustrated by scaling the database or by changing the access pattern from uniform access to various degrees of skewed access to a database.

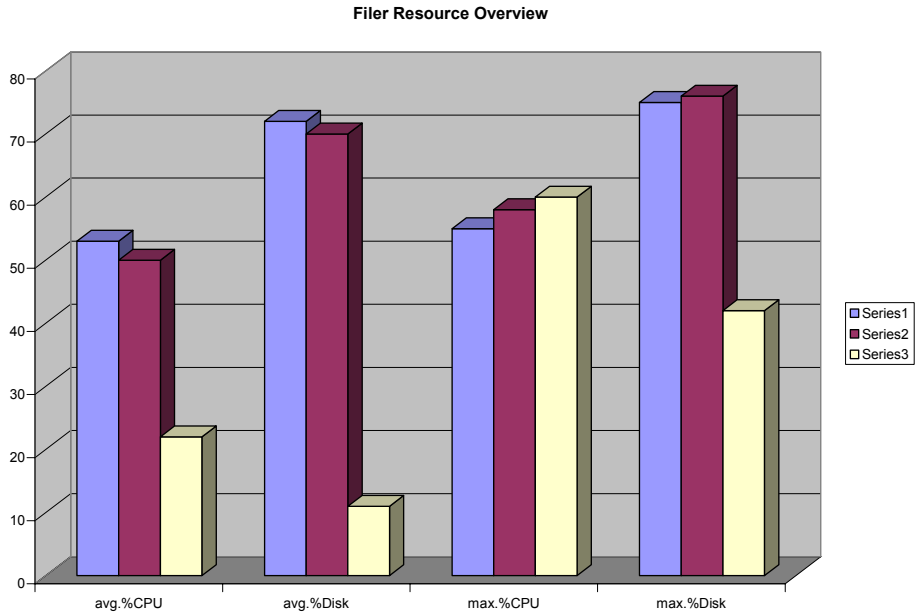

Figure 5) Effect of buffer cache size (utilization).

Analyze a NetApp storage appliance's resource use, then relate it to SQL Server's cache hit ratio. The cache hit ratio might change when consolidating, as the throughput increases, or as the database size grows; the change might show up as a changed access pattern on the storage device.

2.1.1. Correlating the Buffer Cache Size with the Database Size

The buffer cache optimizes physical I/O in several ways:

Frequently referenced pages will only have to be read once from permanent storage
A single database page can be updated multiple times before it is actually written to disk
Database checkpoints make it possible to flush database pages to disk much more efficiently than if one page had to be written at a time; pages are sorted according to file offset
Reading database pages directly from the buffer cache makes SQL Server's I/O path very efficient

A large buffer cache might also create a potential I/O bottleneck by flushing too much data to permanent storage in too short a time. For example, if the buffer cache is 4GB and the checkpoint occurs whenever 75%[2] of the cache contains updated database pages, then the database engine has to write 3GB of data as quickly as possible, which might create a response time problem for the user of the database; the solution is to execute more frequent checkpoints.

SQL Server does everything possible to read and write database pages as efficiently as possible by sorting and prefetching database pages according to their location in database files. Read is minimized by keeping previously read database pages in the buffer cache as long as possible. Section 2.1 illustrated how SQL Server's buffer cache indirectly affects the physical I/O to database files, and it is important to understand what can be deduced from this relationship

[2]Checkpoint intervals are indirectly controlled by the database's "recovery time" option.

between a given cache size, database size, and the database access pattern. If the buffer cache size is 2GB and every database page is uniformly accessed, then the cache can contain any of the following:

100% of a 2GB database
20% of a 10GB database
2% of a 100GB database

Only rarely will any database application uniformly access every page in all database tables; mostly only part of a database will be references, with some skewed access; some pages are referenced more frequently than other pages. If an application frequently accesses 10% of a database, then the 2GB cache can contain any of the following:

100% of frequently accessed pages; 20GB database
50% of frequently accessed pages; 40GB database
20% of frequently accessed pages; 100GB database

Because of skewed access, some database pages will be referenced often enough to keep them in the buffer cache for future access, resulting in a high hit ratio.

MB Buffer Cache	Percent Buffer Hit Ratio
50	71
150	78
500	99

Figure 6) Buffer hit ratio (section 2.1 example).

Figure 6 and the discussion in section 2.1 illustrate how the buffer hit ratio will directly influence the physical I/O that the storage device has to be able to support. The buffer cache will by default be allocated dynamically by SQL Server. The actual size depends on available server memory; available memory depends on the number of SQL Server instances and other processes utilizing the memory.

An OLTP type of database ought to have a cache hit ratio of 80% or better, which is very different from a data warehouse type of database, which will often have a hit ratio that is less than 50%. Of course, it all depends on the size and access pattern of the database.

2.2. Physical I/O and I/O Functions

SQL Server 2000 uses several access methods supported by Microsoft® Windows 2000 to improve its disk I/O performance. For a description of the physical structure of SQL Server databases, see section 2 in TR3323: SnapManager® for Microsoft SQL Server 2000 Best Practices, which also discusses database layouts when SnapManager for SQL Server is used for backup and restore.

Access methods used by SQL Server 2000 and supported by Windows are important background information to understand SQL Server's I/O functions. I/O functions used by SQL Server are:

Scatter-gather I/O
Asynchronous I/O
Synchronous I/O

2.2.1. Scatter-Gather I/O

Scatter-gather I/O allows a read or a write to transfer data into or out of discontinuous areas of memory in 8kB chunks. If an instance of SQL Server 2000 reads in a 64kB extent, it does not have to allocate a single 64kB area and then copy the individual pages to buffer cache pages. It can locate eight buffer pages and then do a single scatter-gather I/O specifying the addresses of the eight buffer pages. Windows 2000 places the eight pages directly into the buffer pages, eliminating the need for the instance of SQL Server to do a separate memory copy.

A database page is 8kB, and a table extent is 64kB (eight pages).

SQL Server sorts read requests according to offsets in the data files, expecting this will deliver the lowest latency.

Table scans are very efficient in SQL Server 2000. The 8kB index allocation map (IAM) pages in a SQL Server 2000 database list the extents used by tables or indexes. The storage engine can read the IAM to build a sorted list of the disk addresses that must be read (or written). This allows SQL Server 2000 to optimize its I/O requests as large sequential reads that are done in sequence based on disk location. SQL Server 2000 issues multiple serial *read-ahead requests at once for each file involved in the scan*. SQL Server 2000 Enterprise Edition dynamically adjusts the maximum number of read-ahead pages based on the amount of memory present and the *responsiveness of the storage subsystem* and is able to read multiple megabytes in a single I/O operation.

The SQL Server 2000 Enterprise Edition advanced scan feature allows multiple tasks to share full table scans. If the execution plan of a SQL statement calls for a scan of the data pages in a table, and the relational database engine detects that the table is already being scanned for another execution plan, the database engine joins the second scan to the first, at the current location of the second scan, by doing the following steps:

1. The database engine reads each page once.
2. The database engine passes the rows from each page to both execution plans.
3. The database engine continues until the end of the table is reached:
    a. The first execution plan has the complete results of a scan.
    b. The second execution plan must still retrieve the data pages that occur before the point at which it joined the in-progress scan.
4. The scan for second execution plan then wraps back to the first data page of the table and scans forward to the point at which it joined the first scan:
    a. Any number of scans can be combined in this way.
    b. The database engine will keep looping through the data pages until it has completed all the scans.

SQL Server might use scatter-gather to read ahead a maximum of 32 extents (2MB of data) when scanning a table, depending on the table's allocation and the storage device's responsiveness.

2.3. Asynchronous I/O

When asynchronous I/O is used, an application that requests a read or a write operation from Windows 2000 will immediately return control to the application. Using asynchronous I/O allows instances of SQL Server to maximize the work done by individual threads by processing other user requests while physical I/O is done in the background.

Most of SQL Server's I/O operations are executed asynchronously. SQL Server will initiate multiple asynchronous read requests during a table scan; it will read from each data file a table is spread across. All write requests from the buffer pool are executed asynchronously.

2.3.1. Writing Database Data Pages

Changes to database tables are written asynchronously in the background without directly influencing processing of active transactions. Users' response times can be influenced indirectly by write activities in several different ways. If the storage device cannot keep up with the rate of writes of data pages, then SQL Server might run out of free buffer pages. If that happens, then each read has to wait for a free buffer page, and that will slow down every user's response time and decrease the system's total throughput. Concurrent reads and writes from the same storage device might influence each other's latencies, but this is usually not a problem with NetApp storage.

Figure 7 illustrates an example when the storage manager will create a scatter-gather write sorted according to each page offset from the start of the file. The two adjacent pages (2031 and 2032) will be processed as a single I/O operation.

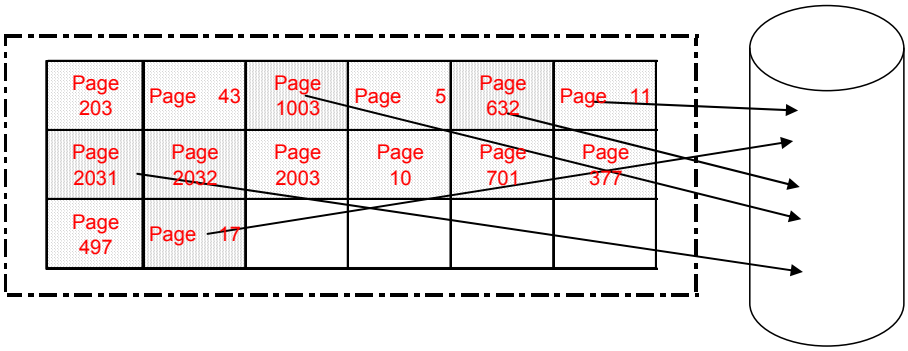

Figure 7) Writing buffered data pages.

2.3.2. Writing to File Groups

A file group is a logical grouping of physical files. SQL Server uses file groups as a type of I/O load balancing object. Tables are created on a file group, and each table extent is created in a round robin fashion that spreads the table in 64kB extents across all files in the file group. When a table runs out of preallocated table space, then SQL Server allocates another extent (eight database pages, 64kB) that will be allocated in the next file in the file group.

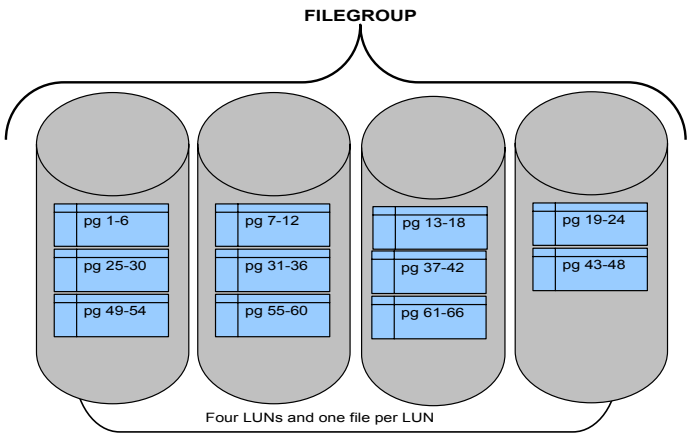

Figure 8) Writing to file groups.

Figure 8 illustrates a file group with four files. One table extent is created at a time. The first extent (pp. 1–6) is allocated on the first file, the second extent is allocated on the second file, and so on until no more file space is left, at which point the physical files will be extended if possible.

SQL Server uses this allocation scheme when scanning a table by issuing multiple 64kB I/O requests concurrently across all files in a file group. SQL Server will also use this information when scatter-gather reading multiple extents.

Use of file groups makes it easier to manage large databases, and SnapManager for SQL Server makes it easy to move one or more files to another NetApp volume, which is very useful when a database grows too large to be contained in a single volume/filer because of space or performance limitations.

2.3.3. Automatic Checkpoints

SQL Server 2000 always generates automatic checkpoints. The interval between automatic checkpoints is based on the number of records in the transaction log, not time. The time interval between checkpoints can be highly variable: it is long if few modifications are made in the database and short if a lot of data is modified.

The interval between automatic checkpoints is calculated from the **recovery interval**, which is a server configuration option. This option specifies the maximum time SQL Server should use to recover a database during a system restart. By default the recovery interval is zero, which implies a crash recovery of one to two minutes.

The recovery interval is in many cases too short, so short that the database will continually be in checkpoint mode, which is not very efficient. Remember that a short recovery interval will create frequent checkpoints with short peak write loads (great for databases supporting low transaction rates), and high values will create longer write peak loads during checkpoints; how long depends on the buffer cache size. The effect of changing the recovery interval value has to be evaluated in relationship with the I/O load generated by user transaction activities and how the storage device handles the I/O load.

2.3.4. Reading from Data Files

If a relatively large buffer cache is retained in memory, an instance of SQL Server can significantly reduce the number of physical disk reads required per transaction. A frequently referenced page will most likely be read only once from disk into the buffer cache and will probably remain there, eliminating further reads. The number of database pages referenced by a SQL Server instance will heavily influence the I/O access pattern. Small databases might be cached completely in the SQL Server memory buffer, eliminating future reads. Skewed access to very large databases might also cache most, if not all, referenced pages and thereby change the read and write ratio to almost only writes with a small ratio of reads; see section 2.1.1.

2.4. Synchronous I/O

In synchronous I/O, the operating system does not return control to the application until the read or write completes. SQL Server 2000, like other relational database engines, uses a write-ahead log, ensuring that no database page is written to storage before the associated page is written to disk.

Writes to a transaction log are synchronous when transactions are committed. SQL Server has to flush all log records associated with a committed transaction before the application can start the user's next transaction.

2.4.1. Writing the Transaction Log

To achieve maximum throughput, the log manager maintains two or more log caches; one log cache is currently used for new log records. A log cache can be in one of two possible queues: the flush queue or the free queue. A log cache is placed in the flush queue when a user thread, for example, commit a transaction, requests it to be written to disk; a log cache is placed on the free queue after it has been flushed to disk.

Figure 9 illustrates a SQL Server environment with five log caches. Two caches are located in the flush queue; users who are waiting for a cache to be flushed cannot continue processing current or pending transactions until all previous log caches have been flushed to disk.

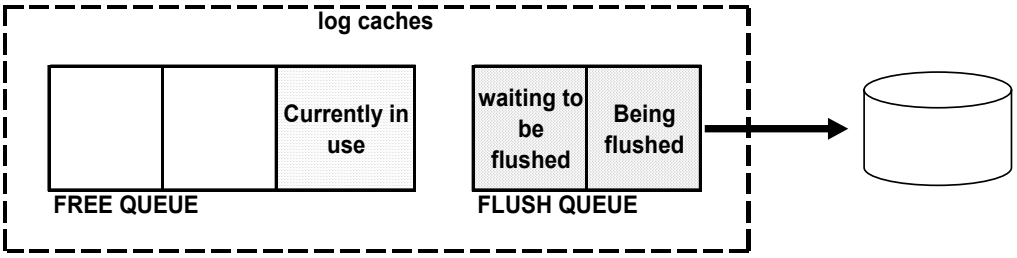

Figure 9) Log cache overview.

The size of the write block to the transaction log depends to a great degree on the rate of transactions modifying the database. The smallest write size is 512 bytes, but that size will increase with the load and number of users—how large is difficult to say, but 1kB, 2kB, 4kB, and even 16kB are not unusual.

If flushing the log cache takes too much time, then on average more users will be waiting in the flush queue, and users' response times will increase.

Flushing of the transaction log has not been observed to be an issue with NetApp storage.

3. Optimizing NetApp Storage for SQL Server

NetApp storage appliances allow SQL Server to efficiently service database operations, such as transactions, backup, database verification, and database restore. Disk throughput is a limited resource that can deliver a given number of operations per second, depending on the access pattern.

Writing is managed by NetApp storage very efficiently, acknowledged as soon as recorded safely in NVRAM and flushed to disk asynchronously during checkpoints. The factor that limits throughput is often the number of disk drives in the accessed volume.

3.1. Read-Ahead and Setting of minra

Blocks frequently read from NetApp storage might stay in filer memory for future reads after the initial read. Database applications with sequential read access will often retrieve data from NetApp filer memory because of NetApp read-ahead technology.

The volume option minra controls whether the NetApp storage device will read ahead. If minra is on, then the storage device will only read the database pages requested by SQL Server. If minra is off, then the storage device will speculatively read ahead, depending on the access pattern. Read-ahead might benefit SQL Server's performance, but it depends on the I/O load generated by an application's specific environment.

Use of the minra volume option depends on the physical access pattern generated by the application in the specific system environment. Databases with sequential access, for example, data warehouse, will benefit from minra set to off, and databases with random access, for example, OLTP, will benefic from minra set to on.

3.2. FCP or iSCSI Infrastructure Setup

Many performance issues are related to poorly configured network infrastructure supporting the chosen I/O protocol. Therefore, correct design and configuration of the network are critical for optimal performance.

Set up and configure FCP or iSCSI according to recommendations in the guide for SnapDrive™ 3.1 (NOW access required) at Installation and Administration Guide for SnapDrive.

3.2 Files, File Groups, and Disks

When translating SQL Server storage requirements into NetApp storage requirements, several storage-related factors have to be considered in two different but related areas: space and throughput. Before discussing these factors, let us look at basic database layout considerations.

SQL Server stores data and the transaction log in disk files. As a default, data and log files are created in the location specified in the server configuration.[3] However, to maximize performance and manageability, a few basic principles should be applied:

Spread data over as many disks as possible. In general, the more disks (spindles) you have (regardless of their individual size) and the faster your access to them, the more quickly the storage engine can read and write data. The larger your system usage becomes, the more important it is to separate data files from log files by storing them on different sets of physical drives.

Placing log and data files in the same LUN and volume has not been observed as a performance issue with NetApp storage.

Use file groups to make your enterprise database more manageable.

Every database begins with one default file group, which can work effectively without additional file groups, and many systems will not need to add additional file groups. However, as a system grows, the use of additional file groups can provide more manageability, when implemented and maintained by a qualified DBA. Multiple file groups make it possible to place tables with very different access patterns on different storage devices. Hence, when implementing or optimizing a database design, the database administrator needs to consider the configuration of the database

[3]The model system database is the template for creating new databases.

storage components, particularly the layout of physical and virtual disks and the arrangement of the database files across disks.

3.3.1. Space Considerations

Database files are placed on LUNs, which are created on volumes. A volume is the main space consideration when creating LUNs for placing database files. Space is needed not only for current databases' files but also for the following:

Volume space for Snapshot™ backups—free volume space equal to the total space allocated to all LUNs placed in the volume
Space required for each new Snapshot copy equal to all volume blocks changed since the previous Snapshot copy was created
Backup of a transaction log, requiring space in SnapInfo for the active section that is copied to that directory
Space for metadata created by SQL Server and SMSQL during backup creation
Space in SnapInfo where copies of system database files will be streamed during backup
Space to expand LUNs as free space is being allocated by growing databases

It is not complicated to estimate volume sizes when current database sizes, change rate, and growth rate are known, but partial knowledge is often the case, which increases the risk.

3.3.2. Throughput Considerations

Throughput requirements are much more difficult to estimate, not only because it is very difficult to understand the access pattern, especially over time as a database is growing, but also because an online database is not the only user of physical I/O; other related operations will also create physical disk loads. Some examples:

When SMSQL requests SQL Server to back up a transaction log, SQL Server will extract log records from the transaction log file and sequentially write extracted data to the dump file in SnapInfo.
Snapshot copy creation will create a slightly noticeable load on the filer.
Verification on a backed-up database will create a read load on disk drives containing the volume. The load has two phases:

- Mounting the LUN in the Snapshot copy as writable will create a light load for a short duration.
- Verifying the database implies reading all database pages and will create a physical read load on the filer.

It is recommended to offload the verification process to a dedicated server, which will decrease resource use on the online server.

If SnapMirror® is used, each replication creates a read load on the source volume(s).

- It is recommended to place all database files and SnapInfo in a single volume.

Restoring databases will restore all LUNs used by the databases in three phases:

- Single-file-SnapRestore® (SFSR) will be used to restore all LUNs used by databases to be restored. This action will create disk activities.
- SMSQL will restore the full backups in the LUNs. This will create limited read/write activities.
- SMSQL will ask SQL Server to apply backed-up transaction logs to recovering databases and will sequentially read backed-up transaction logs located in SnapInfo and subsequently flush database pages to disk.

4. Sizing NetApp Storage for SQL Server

Sizing high-end systems for both space and throughput is best done when all parties are cooperating to create the correct storage infrastructure, fulfilling both space and throughput[4] requirements. The sizing process is done in two related phases. The first phase involves collecting information related to the current database environment: number of SQL Server instances, databases managed by each instance, current database sizes, growth rate, and rate of change. This information is used during phase 2 to decide filer model, number of disk drives, volume, and LUN configuration. The two phases are described in section 2 of TR3323: SnapManager for Microsoft SQL Server 2000 Best Practices.

As has been illustrated throughout this technical report, throughput requirements depend on many interrelated factors that make it difficult to size storage for throughput; the actual load depends on the application utilizing SQL Server databases, number of users, mix of transactions, hardware configuration on which SQL Server is running, resulting physical I/O load, and access pattern. Still, we can approximate expected throughput requirements from what is known about the current SQL Server environment and how it relates to the new setup.

Customer sizing requirements might vary greatly depending on each customer's situation, but in general the situations can be grouped into three different categories; each category might require special consideration.

Consolidate local storage to NetApp storage; current SQL Server instances will not be consolidated. Since the SQL Server instance will continue using the current Windows server, the effect of SQL Server's buffer caching will not change, and the I/O rate should not change either. Therefore, *sizing NetApp storage is based on current database storage requirements as described in section 4.1.*

Consolidate multiple SQL Server instances to fewer servers and consolidate databases to NetApp storage. As discussed throughout this paper, SQL Server optimizes its physical I/O by caching frequently accessed data as well as possible, since the resulting physical I/O depends to a large degree on the caching hit ratio. Because of this fact, it is important to monitor each SQL Server instance's buffer cache hit rate before and after consolidation (Figure 10). The goal is to end up with a cache hit ratio as good as or better than after migration.

If the cache hit ratio becomes lower after consolidation, then the physical I/O will be greater, and the access pattern might also change (see section 2.1).

Figure 10 shows how perfmon can monitor SQL Server's buffer cache hit ratio.

[4]Throughput is used as a synonym for both megabytes of I/O per second and time to servicing the I/O requests.

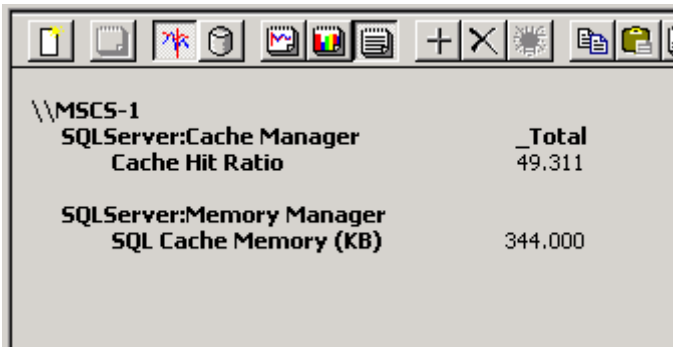

Figure 10) Perfmon buffer cache hit ratio.

When it is a completely new application, the customer may not know the storage requirements for the environment. Sometimes the user may have an idea about the load—for example, that the transactions load will be 100,000 or the initial database size will be 10 million rows—but this type of information is useless. Let us analyze why this information cannot be used for sizing storage requirements.

It is better to consider the response to "How much storage does a table with 10 million rows require?" The smallest size of a row is one byte, and the maximum size is 8060 bytes, which partly explains why the number of rows cannot be used for sizing storage requirements. It is impossible to estimate a space requirement without knowing the row size, but the actual space requirement also depends greatly on how a table is designed. The example in Figure 11 is a good example of why it is important to know the table layout and the space occupied by a single row.

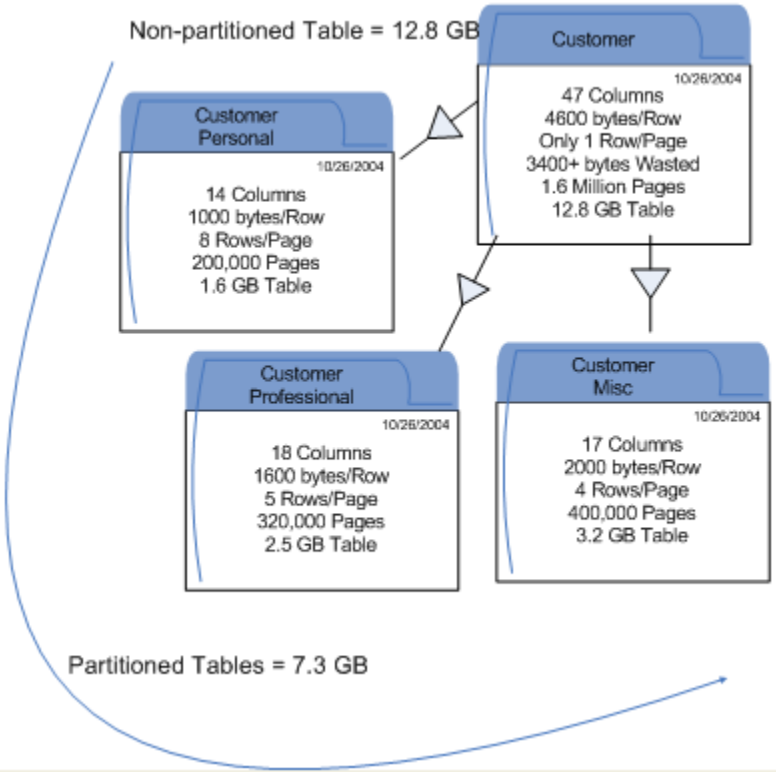

Figure 11) Effect on storage requirement when partitioning a table.

Figure 11 compares space requirements when a single customer table is used. The summary for when the table has been partitioned the summary is as follows:

Each row size in the nonpartitioned table is 4600 bytes. A database page is 8kB, and since a row cannot span more than one page, in this case each page can only contain one row; 3400 bytes are wasted.

The nonpartitioned table not only will take up more physical storage but also will use more buffer cache; the results will be a lower cache hit ratio and a higher physical I/O rate.

The partitioned table uses less physical storage and less buffer cache, which will result in a higher cache hit ratio and less physical I/O.

Sizing storage requirements in this case is very complicated and can only be done in cooperation with the database administrator and maybe the application vendor. If the database is installed, then it is possible for the DBA to calculate row sizes with the help of Enterprise Manager and thereby estimate space requirements for each table.

The number of transactions cannot be used for sizing storage requirements because a transaction might manipulate data from a single row or many rows from one or many tables. If the transaction is known, then it is possible to estimate the number of logical I/O operations required. If the sizes of the database, access pattern, and buffer cache are known, then it is possible to estimate the number of physical I/O operations required per transaction. These calculations are complicated and should only be done by the DBA responsible for the new environment. Also, it is important to know the ratio of reads and writes.

4.1. General Sizing Recommendations

The recommendations in this section are based on TR3323: SnapManager for Microsoft SQL Server 2000 Best Practices (section 3). Sizing storage requirements for a specific SQL Server environment consist of several related objects:

- System databases
  - Focus on **tempdb**
- User-defined databases
  - Data files
  - Transaction log files
- SnapInfo
  - Metadata
  - Full backup
  - Backup of transaction log
  - Streaming-based backup of system databases
  - Streaming-based backup of user-defined databases
- Snapshot copy creation

SQL Server tries to optimize its physical I/O as well as possible by caching database pages and by writing modified cached database pages to permanent storage in a sequential fashion, as described throughout this technical report. Hence, even OLTP-type applications will try to read and write sequentially.

4.1.1. Space Requirements for System Databases

All system databases are fairly small, less than 20MB. Limited changes take place on system databases (not including tempdb), unless an application uses a system database for some purpose, so they only have to be backed up whenever changes are made. Tempdb is recreated every time SQL Server is restarted and should never be backed up, because the backup will never be used.

It is recommended to place all system databases in a dedicated LUN. The I/O load on tempdb can become very high (how high depends on the application, number of databases, and SQL statements executed by the SQL Server instance).

An overview of space requirements for system databases is provided in Figure 12. Since streaming backup (SnapManager for SQL Server 2000) is used for all system databases, the actual space requirement for SnapInfo (SnapManager for SQL Server 2000) is equal to the size of the online database, since the system databases are copied to SnapInfo. Tempdb is recreated every time SQL Server is restarted; therefore, there is no need to back up tempdb. Space and throughput that tempdb must be able to support depend on the specific environment.

System Database	Online Space	SnapInfo Space	I/O Rate
System databases (without tempdb)	Less than 150MB	300MB	Low and not an issue
Tempdb	Environment dependent	No backup	Environment dependent; can be very high

Figure 12) Space requirement for system databases.

4.1.2. User-Defined Databases

Space and throughput requirements for user-defined databases depend on measurements collected at peak loads of the production system.

Calculating space requirements for user-created databases requires several measures, as shown in Figure 13, which illustrates space requirements that have to be collected. Figure 14 shows throughput data that has to be collected. Space requirements are used as an estimate for how much storage space is required, and throughput requirements are used to estimate the number of disk devices necessary for supporting the required I/O rate.

SQL Server Instance	Database	Database Size	Database Type
SQL Server 1	DB 1	39GB	DSS
	DB 2	25GB	OLTP
SQL Server 2	DB 3	200GB	OLTP
	DB 4	75MB	Test
	DB 5	50MB	OLTP
2 SQL Server Instances	**5 databases**	**265GB**	

Figure 13) Sample current SQL Server information.

Database	Reads per Second	MB Write per Second	Change Rate per Month	Growth Rate per Month
DB 1	5MB	1MB	15%	5%
DB 2	8MB	1.5MB	10%	8%
DB 3	20MB	6MB	12%	6%
DB 4	0.5MB	0.1MB	5%	40%
DB 5	1.5MB	0.2MB	20%	10%
Total	**35MB**	**8MB**	**32GB**	**16GB**

Figure 14) Sample I/O throughput at peak load.

Figure 15 describes the data to be collected and the tool to use to collect the data. Initial database size and growth rate are fairly easy to collect, whereas change rate is more difficult to measure.

User-Created Databases	Initial Space	Growth Rate per Month	Change Rate per Hour
Data Files	Total physical size of database files, usually measured in GB.	Growth rate of physical files used by database, usually measured in MB.	Database pages changed between backups will use space in Snapshot copies.
Transaction Log Files	Will stabilize at a given size depending on configured	Will only grow in the beginning until a balance reached between arrival rate of	Each time transaction log is backed up, log file will be logically truncated and made available for new

	recovery mode and how often log is backed up.	new transactions and frequency log is backed up.	transactions. Physical file size will not change.
How to Collect	Figure 8 illustrates how to find a database's current size.	Figures 9 and 10 show how to use perfmon to monitor size of a database's files.	There is no easy methodology to collect this information. Understanding the application to create educated guesses is the best way.

Figure 15) Space requirement for system databases.

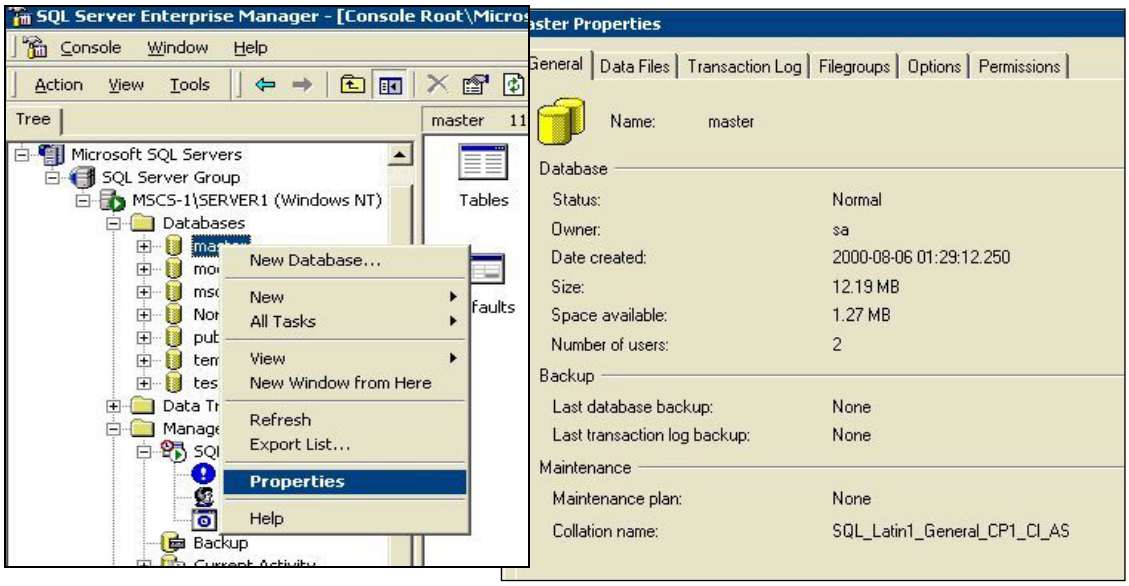

Figure 16) How to find database size with Enterprise Manager.

Figure 16 shows how to find a database's current size. **Size** is the current preallocated physical database size. **Space available** is the current free preallocated space available for new inserted database objects. The **Data Files tab** will show the number of data files and file groups and the size of each data file. The **Transaction Log tab** will show log files and size.

Figure 17 illustrates monitoring the size of the **test** database's data files and transaction log file. **Log Growths** is the total number of times the physical size of the log file has grown; this value should stabilize after a while as a balance is reached between arrival rates of new transactions that modify data and the rate of transaction log backup.

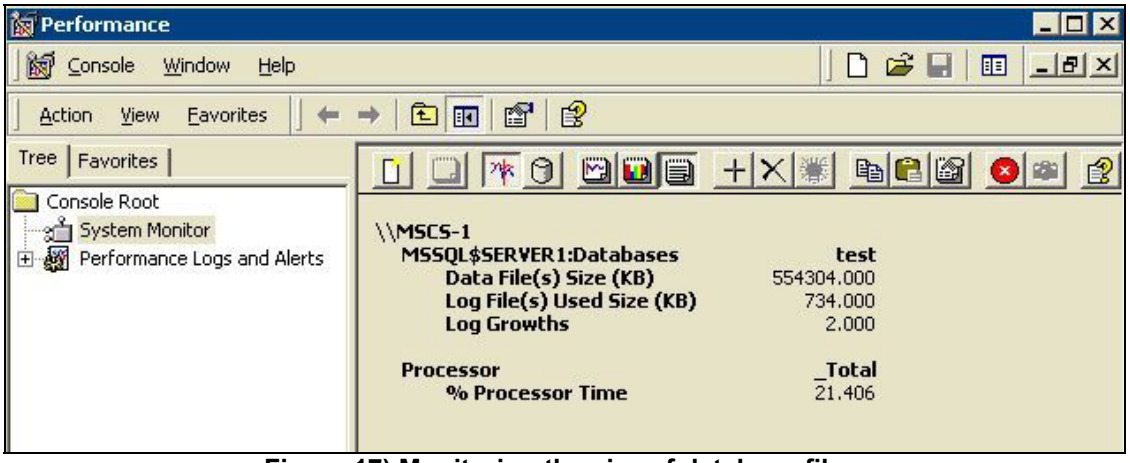

Figure 17) Monitoring the size of database files.

4.1.3. Sizing a Volume for Space Requirements

With the data collected, the space requirements can be calculated in two steps:

1. LUN size = for each database (<initial database size> + <growth rate * months>)
2. Volume size = for all LUNs in volume (<LUN size * 2> + < change rate (adjusted to time between Snapshot copy creation) * number of Snapshot copies>)

Step 1 calculates the space requirements for all databases using a LUN including the required space for growing the database files over a given period—for example, six months. Step 2 calculates the volume size required for all LUNs, including Snapshot copies for the same period.

4.1.4. Sizing a Volume for Throughput

An application's I/O performance depends on its I/O load's characteristics and number of physical disk drives supporting that I/O load. A disk drive can only support a maximum number of I/O requests per seconds; how many it can support depends on access pattern and how fast the disk is.

Current 10,000 RPM and 15,000 RPM disk drives can support roughly 130 to 300 and 180 to 400 4kB read or write operations per second per disk, respectively. Therefore, it is important to estimate the I/O rate for all LUNs utilizing the same volume.

When estimating total peak throughput requirements, remember to include:

Other processes using the same NetApp storage device
SnapMirror processes
Backup of transaction logs (Snapshot backup will not influence the throughput)
Database verification

When the throughput requirements are well understood, then calculate the number of disk drives required to support the required throughput. Current disk technology can support a maximum number of I/O requests per second; how many it can support depends on the access pattern and how fast the disk is. Current 10,000 RPM and 15,000 RPM disk drives can support roughly 130 to 300 and 180 to 400 4kB read or write operations per second per disk, respectively. Therefore, it is important to estimate the I/O rate for all LUNs utilizing the same volume.

4.1.5. Sizing Transaction Logs

Access to a database's transaction log is sequential. How high an I/O load the transaction log must be able to support can be estimated by monitoring the current I/O load with the Microsoft performance monitor tool (perfmon). Figure 18 is an example of monitoring a test database's transaction log. Perfmon has enough data to size for both space and throughput.

It is in general a performance advantage to place files with different access patterns on different volumes (as described in technical report 3283), but in many cases the performance requirement is not high enough to make the advantage noticeable. Hence, in most cases all database files can be located in the same LUN/volume. When performance is very critical and the I/O load is very high, then it is recommended to place the log file in a LUN separate from the LUN used by the data files and on different volumes. Hence, even though the access to a transaction log is sequential, the sequential nature will change to somewhat random when multiple databases' log files are placed in the same LUN/volume.

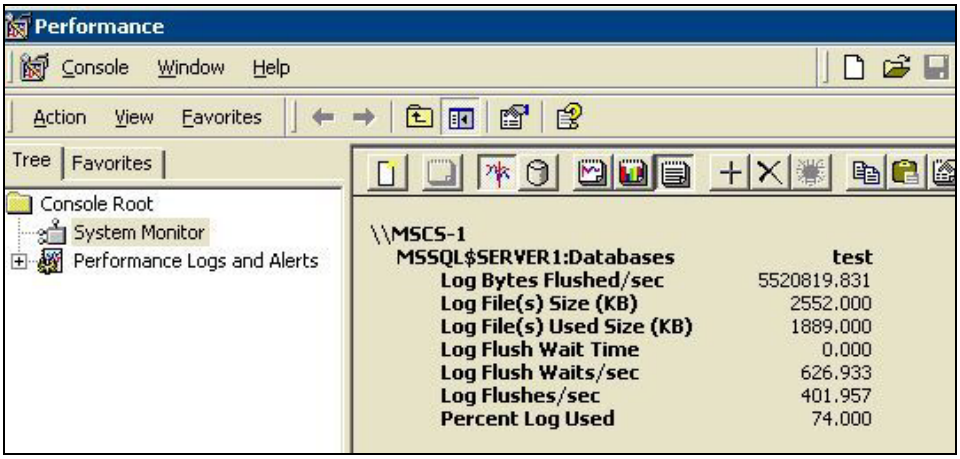

Figure 18) Monitoring transaction logs.

4.1.6. Sizing SnapInfo Used by SnapManager for SQL Server

Space requirements for SnapInfo depend mainly on the number of backed-up transaction logs and whether streaming backup is used[5]. Figure 19 is an overview of actions taken when a backup is executed. Any backup action will create metadata in SnapInfo; metadata is created by both SMSQL and SQL Server 2000.

Action		Outcome			
	Backup Type	**Metadata created**	**Streaming Data**	**Snapshot Copy of Volume with Database**	**Snapshot Copy of SnapInfo**
	Full Backup of User Database	x		x	x
	Full Backup of System Database	x	x		x
	Backup of Transaction Log	x	x		x[6]

Figure 19) SnapInfo content overview.

Space requirements for SnapInfo depend on all of the above factors, including the number of databases, size of streamed databases, number of backups to be kept online, and active size of a database's transaction log since the last time the transaction log was backed up. Figure 20 is an overview of what influences SnapInfo space requirements.

	Space Requirements	**Comments**
Metadata	Less than 20kB	
Streaming Full Backup	Equal to size of online database	Initial sizes of system databases less than 20MB; see Figures 8 and 9
Snapshot Copy	Equal to changed filer blocks since creation of last Snapshot copy	
Transaction Log Backup	Equal to active sections of transaction log	Figure 10 illustrates how to monitor used size

Figure 20) SnapInfo space overview.

4.1.7. Snapshot Management

When sizing volumes containing virtual disks for Microsoft SQL Server 2000, consider the

[5]Streaming-based backup will always be used with system databases. User-created databases sharing LUNs with a system database will also use streaming-based backup, which is not recommended.

[6]A Snapshot copy of the volume containing SnapInfo will be created by default, but can be turned off.

number of Snapshot copies that will be created and the length of time they will be kept. Disk space consumed by a Snapshot backup consists only of blocks that have changed since the last Snapshot copy was created.

For example, a Snapshot copy of a 100GB database is created at midnight. If between midnight and noon the following day, 500MB of data is changed or deleted from the database, the copy will consume only 500MB.

However, if that same 100GB store Snapshot copy is kept for 20 days and experiences the same rate of data change during that period, the Snapshot copy will now consume **10GB** (500MB * 20 days). Properly managing Snapshot copies requires balancing the number of copies and the length of time they are kept.

4.1.8 Space Reservation

As part of the space reservation feature, WAFL® reserves blocks equal to 1X times the specified size of the LUN being created. If a volume of 80GB is being created with a planned growth rate of an additional 24GB, enough spindles to accommodate **208GB** ((80 + 24) * 2) will be required. Space reservation is designed to guarantee writes for virtual disks. More information on space reservations can be found later in this document.

4.2. Volume Sizing Recap

Figure 21 is an overview and an example of estimating space requirements. This example does not size for throughput, but some discussion is included during the description of the content.

*A link to the Excel document is the **SQL Server space calculator**. The spreadsheet is a template sizing storage requirements for system databases, five user databases, and SnapInfo. The spreadsheet calculates the LUN and volume sizes and the database size after x months.*

Filer	Volume RAID-DP™ Data + Parity	Used Volume Space Including Snapshot Copies (GB)	Database Name	LUN Size + 10% (GB)	Drive Letter	Database Size after X Months	Initial Size (GB)	Growth Rate per Month (%)	Change Rate per Month (%)	Size for Number of Months	Transaction Log (GB)	TempDB (GB)	System Databases (GB)	SnapInfo (GB)
1	16+4	638	OLTP 1	**184**	I:	167	150	5	15	6	0.5			
			OLTP 2	**15**	J:	14	10	20	25	6	1			
			OLTP 3	**3**	F:	3	2	15	10	6	0.2			
			OLTP 4	**1**	F:	1	0.7	15	5	6	0.2			
	8+2	692	DSS 1	**238**	K:	217	200	2	10	6	0.5			
	8+2	376		**144**	L:	131	100	25	5	6	0			
	6+2	99		**45**	T:							15		
	6+2	444		**202**	S:								2	200

Figure 21) Space allocation overview (Excel spreadsheet).

The information in Figure 21 is an example of a recommended layout but is only one solution out of many possible layouts. Recommendation must be related to the specific environment and its requirements.

The columns are defined in the **SQL Server space calculator**.

4.3. Special Sizing Considerations for MSCS

If the deployment will use MSCS, then special consideration needs to be taken when sizing storage requirements. Because of the fact that the disk drive letters are shared by all nodes in a system, the number of possible LUNs may be exposed to an issue with MSCS due to the fact that drive letter X needs to be available on all nodes in case of failover. This has implications for the total number of LUNs that can be created on an MSCS configuration and is important, because multiple SQL Server instances are usually active in an MSCS configuration. The total number of drive letters available for a clustered configuration is generally 23. Therefore, two nodes will each have 11 possible LUNs available plus the quorum disk. Four nodes (Windows 2003) will only have five LUNs available per node plus the quorum disk.

Each SQL Server instance active on a node requires three LUNs:

SQL Server and its system databases
User-created database
SnapInfo (if SnapManager for SQL Server is used)

A large and powerful Windows platform will often support multiple SQL Server instances. Each instance requires three LUNs, two instances require six LUNs, three instances require nine LUNs, and four instances need 12 LUNs.

It is important to remember that after a failover all server resources will be shared by all SQL Server instances running on that server, and this can affect SQL Server's buffer cache hit ratio. When a single SQL Server instance is running on a server, then all available memory might be used by that one instance, but after a failover the available memory has to be shared by all instances running on the server, *and this might completely change the physical I/O rate and access pattern.*

5. Ongoing Capacity Planning and Monitoring

Capacity planning involves planning for the capacity of a system to maintain the level of service expected by the user community and consists of two phases:

Precapacity planning or sizing as discussed in this technical report
Postcapacity planning, which will be discussed in this section

The success of precapacity planning depends on the quality of the data used to size the system, as described in section 5. Sizing is important because a new system will be expected to meet the term of a service-level agreement (SLA), and it is expected that the system configuration and infrastructure will be powerful enough to meet the SLA even during times with peak load.

Capacity planning or predictive analysis involves complex and ongoing performance measurements and study of hardware and software resource consumption. The study is used to predict or project resource consumption to plan for system capacity increases as needed. When more resources are needed, before encountering critical resource limitation, anticipate and plan ahead. When this approach is used, money can be saved and the total customer experience can be improved because users are spared aggravation.

An in-depth discussion of capacity planning is outside the scope of this technical report, but there are many excellent technical reference guides available. One such reference guide that focuses on SQL Server 2000 is from Microsoft: *SQL Server 2000 Performance Tuning* ("the in-depth, practical guide to performance tuning, optimization, and capacity planning," by Edward Whalen, Marcilina Garcia, Steve Adrian DeLuca, and Dean Thompson; ISBN 0-7356-1270-6).

This report will only list performance counters that are candidates to be collected with the purpose of analyzing the trend over time and thereby predicting when it will be beneficial to increase system resources.

Performance monitoring is something every administrator has to do and should do; performance tuning is something everybody has to do periodically, and the same counters can be used for capacity planning purposes. The main goal with performance monitoring is to make certain that no resource is a bottleneck that results in requests waiting for access to the limited resource, whereas capacity planning is to predict when it will be beneficial to add system resources.

This section will focus on one method to set up **perfmon** with a minimum basic set of counters to monitor a system and how to analyze the data.

5.1. Basic Set of Counters

It is not possible to track everything, but the counter described in this section give a good overview of what is happening on the server. This short list of counters could be dropped into Performance Monitor to get a baseline on the performance of all servers. The counters can also be added to a third-party tool.

Process Object → % Processor Time

The measurement is the amount of time that the server is executing a nonidle thread. This counter provides an indication of a possible bottleneck. A server will go to 100% CPU at times, but unless this is sustained for a period of time (for example, 30 minutes, system dependent), it is usually nothing to worry about. However, over time, for example, a month, if the CPUs are averaging constantly more than 50%, it might be a good idea to start planning for an upgrade. If processors are averaging constantly greater than 70%, then it might be time to hurry that planning along.

System Object → Processor Queue Length

Watch the value over a sustained period of time to see if the CPU is a bottleneck. The counter measures how many threads are waiting for processor time. The count includes all threads that are "ready" and not waiting on some other thing such as I/O to complete. The general rule of thumb is that this value should be less than two times the number of processors in the system. If it is growing over time and the CPU % is greater than 80%, more or faster processors are probably needed.

Memory Object → Pages/sec

This counter measures the number of memory pages read from or written to disk. From the SQL Server 2000 Operations Guide:

Defined as the number of pages read from or written to disk to resolve hard page faults. (Hard page faults that occur when a process requires code or data that is not in its working set or elsewhere in physical memory and must be retrieved from disk.) This counter was designed as a primary indicator of the kinds of faults that cause system-wide delays. It is the sum of memory: pages input/sec and memory: pages output/sec. It is counted in numbers of pages, so it can be compared to other counts of pages, such as

memory: page faults/sec, without conversion. It includes pages retrieved to satisfy faults in the file system cache (usually requested by applications) noncached mapped memory files. This counter displays the difference between the values observed in the last two samples, divided by the duration of the sample interval.

This is a counter that cannot be measured independently of a system. It is not possible to say that 1000 pages per second is a high or low value; it really depends on the system. The counter is relative to the system and has to be watched to see if things are changing. Make a change and see if the counter goes up or down. In general, adding memory (or allocating more to SQL Server) should lower this counter, but be careful. Understand this counter's behavior relative to the measured baseline; watch it as changes are made to the system. SQL Server will by default dynamically manage its memory buffer cache, but it is possible to set it manually to a fixed value. If this is done incorrectly too high, then this counter points to the problem.

Memory Object → Available MBytes

This measurement is the amount of megabytes of memory on the computer (from the OS perspective) that is available to be allocated to processes. It's the sum of free, standby, and zeroed memory pages. Look at this to see if the average amount of megabytes is fairly consistent for a baseline perspective. This can often be a clue to some other process or service being added when one expects everything on the server is stable with no new application or workloads.

PhysicalDisk Object → Avg. Disk Queue Length

This counter measures whether I/O requests are being serviced or are waiting in the queue for the disk drive to catch up on requests. This can be a bottleneck in performance for a server in that, if it grows large or is consistently above eight (somewhat system dependent) for a particular disk, then that disk device cannot keep up with the arrival of new I/O requests.

PhysicalDisk Object → % Idle Time

The *% disk time counter* has been analyzed a bit on the Internet, and there are some problems with how the data is collected (see O'Reilly Network: Top Six FAQs on Windows 2000 Disk Performance [Jan. 18, 2002]). The idle time is supposedly more accurate. Subtracting this from 100 gives an idea of how hard the disks are working.

PhysicalDisk Object → Avg. Disk Sec/Write

This counter measures the average time, in seconds, of a write of data to disk. If the average value increases over time, then it points to a potential write performance problem. Possible reasons can be that the load has increased over time, the device is running out of free space, or the ratio of read/write has changed.

PhysicalDisk Object → Avg. Disk Sec/Read

This counter measures the average time, in seconds, of a read of data from disk. If the average value increases over time, then it points to a potential read performance problem. Possible reasons can be that the load has increased over time, the ratio of read/write has changed, the access pattern has changed, or the device is fragmented.

Network Interface Object → Bytes Total/Sec

This counter measures the number of bytes being transmitted per second on the NIC. There will be one instance per NIC on the machine, including one for the loopback adapter.

SQL Server Access Methods Object → Full Scans/Sec

This counter should always be captured. It shows how often a table index is not being used and results in sequential I/O.

From the SQL 2000 Operations Guide: *Defined as the number of unrestricted full scans. These can be either base table or full index scans.* Missing or incorrect indexes can result in reduced performance because of too high disk access.

SQL Server Database Object → Transactions/Sec

This counter measures the number of transactions started per second. Transactions are the basis of everything in SQL Server, and most queries are implicit transactions. This measurement is extremely handy for determining if the load has substantially increased over time. This value is very different from a "business transaction" such as a new order, stock-level, or payment transaction. Each business transaction might consist of multiple measured transactions.

SQL Server Buffer Manager Object → Buffer Cache Hit Ratio

This counter shows the percentage of pages that are found in SQL Server's buffer pool without having to incur a read from disk. A well-balanced system will have hit ratio values greater than 80%. The hit ratio ought to be 90% or better for OLTP-type databases and might be less than 60% for DSS-type databases.

SQL Server Locks Object → Average Wait Time

This counter measures the average amount of time, in milliseconds, that a user is waiting for a lock. Over time it is a good indicator for a general performance problem or if a performance issue is specific to one user.

SQL Server Latches Object → Average Latch Wait Time

This counter measures the average amount of time, in milliseconds, that a latch request had to wait before it was serviced. Over time it is a good indicator for a general performance problem or if a performance issue is specific to one user.

6. Summary

Sizing, performance monitoring, and capacity planning are complex functional areas that are best understood and performed as a team effort and analyzed continually.

The discussions in this paper are intended to be an overview of the process, resulting in right sizing NetApp storage for SQL Server databases and knowing how to predict when to upgrade system components for maintaining system responsiveness, satisfying the service-level agreement (SLA).

The report explains how SQL Server 2000 uses I/O functions supported by Windows 2000 to optimize flushing of database pages to disk. The higher the buffer cache hit ratio is, the less the physical read rate will be. Database pages in the buffer might be changed multiple times between being flushed to disk, which will decrease the physical write rate. Checkpoints make it possible for SQL Server to change a random write pattern to sorted sequential patterns, which are more efficient.

Guidelines for sizing storage for both space and throughput have been discussed, and a supporting sizing Excel spreadsheet for space has been included. Sizing for throughput depends

on the server on which SQL Server is running, available memory for the buffer cache relative to the database size, and the actual physical access pattern generated by the application utilizing SQL Server for storing and retrieving data.

Monitoring is essential for capacity planning purposes but also for verifying the system is correctly sized to service the actual load during peak transaction rate.

7. References and Additional Resources

Part of section 2 builds on information from the MSDN home page, especially: SQL Server architecture.

An excellent guide to SQL Server's internal workings is ***Inside SQL Server 2000*, by Kalen Delancy (ISBN 0-7356-0998-5).**

A basic good capacity planning guide is ***The In-Depth, Practical Guide to Performance Tuning, Optimization, and Capacity Planning*** **(Edward Whalen, Marcilina Garcia, Steve Adrian DeLuca, and Dean Thompson; ISBN 0-7356-1270-6).**

SnapManager 1.0 Installation and Administration Guide (NOW access)

SnapDrive 3.1 Installation and Administration Guide (NOW access)

Network Appliance—SnapManager V1.0 for SQL Server 2000

Network Appliance—SnapManager for Microsoft SQL Server 2000 Best Practices

Microsoft TechNet: Microsoft Server 2000 Resource Kit

An excellent Web site is SQL Server Central.

Network Appliance, Inc.
495 East Java Drive
Sunnyvale, CA 94089
www.netapp.com