

Poor Disk Performance, Time Outs, Database and the SQL Server Errorlog

TechNote Number: SQL-DISK-001

Revision 1.0a

Frank McBath

frankmcb@computationpress.com

Publication Date: 26 December 2007

Abstract

A database essentially consists of two parts: RAM & Disk.

With RAM, it's pretty easy to figure out what the issue is—either you are getting good cache hit ratios or you are not. The fix for memory pressure—add more RAM and build indexes. With disk, that question is often much more difficult to pin down.

As of SQL Server 2000 SP4, informational messages are written out to the errorlog which enumerate stalls in the disk. The disk may be a single disk or an entire storage subsystem. The message coming out of SQL Server when you have the issue looks like this:

```
SQL Server has encountered 4507 occurrence(s) of IO requests taking longer than 15
seconds to complete on file [D:\Program Files\Microsoft SQL
Server\MSSQL\Data\BIGDB_Data.MDF] in database [BIGDB] (7)
```

While the message is “informational”, the fact is your database is having very serious performance issues.

In the end, slow disk = slow database performance.

This paper will try and go beyond the “what it is” and more into how to investigate and fix the actual issue.

What You Are Looking At & What It Means

A modern SCSI/SAS enterprise disk drive gets around 3ms to 8ms disk access times.ⁱ Databases start choking when you get greater than 10ms response times on the transaction log (ideally it should be between 1ms to 5ms).ⁱⁱ When you are getting messages saying “15 seconds” and longer, this means you are more than 149,900% over the expected response time (10ms versus 15,000ms).

While this is only an informational message, the meaning of this event in the errorlog is significant. Slow disk response time can turn into:

- The disk is being thrashed
- Blocking at the database causing very slow user response times, due to slow commits
- Batch jobs running slow, reports being delivered late
- Backups running long into user windows, or worse yet—rolling backups
- Application time outs due to non-response and retries flooding the network
- SQL Server itself stalling out because it can’t flush buffers to disk in a timely way

The last bullet above is particularly important. SQL Server is premised on the Write Ahead Log (WAL) protocolⁱⁱⁱ which states that everything must be written to the transaction log before the data file.

The following diagram shows the chain of events from the CRUD operation, its relationship to the disk drives, and how the IO bottleneck materializes in a slow database:

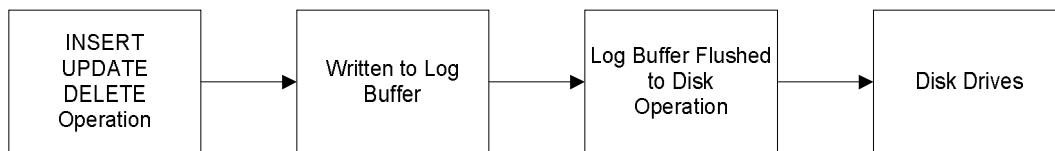


Diagram 1: High level overview of CRUD operation to disk drives.

If the disk does not respond in a timely way, the log buffer does not get flushed fast enough, which in turn means it does not accept anymore data, which in turn means the database cannot accept anymore new CRUD (Create/Update/Delete) operations.

Due to slow disk, SELECT statements doing reads do not work well because the disk can’t service the request and CRUD do not work well because the buffers can’t flush fast enough. The result—the database engine takes it’s foot off the gas pedal and the whole database instance comes to a crawl.

The outcome of a “slow” database is more help desk calls and people upgrading their database server needlessly because they think it will make it faster.

What This Looks Like & Where to Look: Error Logs

In the SQL Server errorlog, you will get entries that look like this:

```
2007-12-05 00:00:20.89 spid64      SQL Server has encountered 4507 occurrence(s) of IO
requests taking longer than 15 seconds to complete on file [D:\Program Files\Microsoft
SQL Server\MSSQL\Data\BIGDB_Data.MDF] in database [BIGDB] (7). The OS file handle is
0x00000484. The offset of the latest long IO is: 0x0000085eb60000
2007-12-05 00:00:36.80 spid64      SQL Server has encountered 40 occurrence(s) of IO
requests taking longer than 15 seconds to complete on file [D:\Program Files\Microsoft
SQL Server\MSSQL\data\tempdb.mdf] in database [tempdb] (2). The OS file handle is
0x00000450. The offset of the latest long IO is: 0x000004b652000
2007-12-05 00:12:41.16 spid3       SQL Server has encountered 1 occurrence(s) of IO
requests taking longer than 15 seconds to complete on file [D:\Program Files\Microsoft
SQL Server\MSSQL\Data\BIGDB_Data.MDF] in database [BIGDB] (7). The OS file handle is
0x00000484. The offset of the latest long IO is: 0x0000081f6b6000
```

From an application error log, you will start getting ODBC and JDBC errors. The following is an example from a JD Edwards application log:

JDEROOT.log HTML client error:

```
19 Dec 2007 12:12:34,946[Line ?][SEVERE][TIERONE01][JDBJ]SQLException
occured in the SQLPhysicalConnection.select(): | Table or View Name =
F4201 | Table or View Name = F42019 | Data Source = Business Data - PROD
com.microsoft.sqlserver.jdbc.SQLServerException: The query was
canceled.com.microsoft.sqlserver.jdbc.SQLServerException: The query was canceled.at
com.microsoft.sqlserver.jdbc.SQLServerException.makeFromDriverError(Unknown
Source)at com.microsoft.sqlserver.jdbc.CancelableRequest.complete(Unknown
```

Additionally, if it is a “soft error” or the like, look at the Windows Event Viewer. Run the following command to see the event viewer run “eventvwr.exe”:

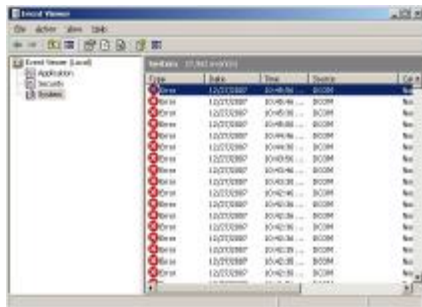


Diagram 2: Event Viewer (eventvwr.exe)

Why Does This Happen

The Microsoft KB article lists a variety of different reasons for this particular error: ^{iv}

- Faulty hardware
- Hardware that is not configured correctly
- Firmware settings
- Filter drivers
- Compression
- Bugs
- Other conditions in the I/O path

It's great if the fix is as easy as putting on a new driver, but experience tends to show it comes down to a combination of a few architectural issues:

- Too few physical disk drives to support the number of IOs being demanded by the workload,
- Not enough IO channels (HBAs) to support the workload which forces a bottleneck (ex. Not enough NICs for iSCSI),
- Poor selection of RAID which either induces a parity bit to be calculated on a write intensive file (ie. TempDB or Log) as in RAID 5 or using RAID 0+1 and cutting the number of physical drives underneath the database to be cut in half which in turn forces less disk to do more work.

How This Error Happens: Poor Performance By Design

The most common problem that we see in the field is "taking all the defaults and leaving it on autogrow". While this may work fine in a small database, it does not scale well in larger systems such as SAP, PeopleSoft, JD Edwards or Siebel.

Common design failures that lead to the disk timing out and other performance issues:

- Buying space not speed.
- One-Big-File theory.
- Everything on one large RAID 5 device.
- Not separating out tempdb, t-log and data.
- Autogrow as management feature.
- Backing up to the same disk that you are reading from.

Everything on One Disk

Some subscribe to the theory that modern hardware can act like a shock absorber for whatever the worst workload you can throw at it: just build a single LUN with disk in some type of RAID configuration and just put the entire database on it. This is just not the case. Thoughtful planning and testing is still mandatory. Mixing tempdb, transaction log and data together can and will cause performance problems. There are other considerations in this, too. For example, by putting log and data on the same physical volumes, you have created a single point of failure. Lose one RAID volume and you have lost log and data. What's more, the performance penalty for calculating the parity bit on the log and/or tempdb can be a significant factor in disk latency (response) times. This can be seen in the DMV's and ::FN tables. (We'll look at this in the monitoring section of the paper.) Effectively when we co-mingle workloads on the same physical disk (mixing log, tempdb, and data)—you are reading and writing to the same disk at the same time which aggravates the situation. For example, you run a SELECT statement with an ORDER BY that is not indexed well. You will be reading off the same disk that you will be writing to with the tempdb. The same can be said for backing up to the same volume that you have your database on: reading/writing to the same disk and a single point of failure (lose the data and lose the backup at the same time). Regrettably, this is a common configuration we see often enough.

The following example shows how tempdb, data, and log are all on the same disk drive:

```
name          db_size
BIGDB         100000.00 MB

name          fileid filename
BIGDB_Data    1          D:\Program Files\Microsoft SQL Server\MSSQL\Data\BIGDB_Data.MDF
BIGDB_Log     2          D:\Program Files\Microsoft SQL Server\MSSQL\Data\BIGDB_Log.LDF

name          db_size
tempdb        3204.94 MB sa

name          fileid filename
tempdev       1          D:\Program Files\Microsoft SQL Server\MSSQL\data\tempdb.mdf
templog       2          D:\Program Files\Microsoft SQL Server\MSSQL\data\templog.ldf
```

Diagram 3: Database Layout

Space Not Speed

A significant issue in designing a system is that people look at a disk subsystem in terms of “space” whereas the real issue are the number of physical spindles to provide speed. While the price of MB/dollar has dropped significantly, the speed of that same megabyte has not increased in years. The end result is the corporate SAN team tries to leverage their “investment” in this technology for the best ROI for the company. Ultimately what ends up happening is multiple applications are shoe horned onto a common disk array, compete for the same physical disk, and all database applications end up having poor IO performance. Note: This strategy can work for file sharing systems. The reason for the poor performance is that all the applications end up on the same physical disk drives even though they may have different logical drive letters. The solution is to break the array up into chunks where one application does not interfere with the other at the physical disk drive layer—easier said than done.

Taking the Defaults

Many people buy into what the sales guy tells them— the TCO story made during the sales pitch told them that an expensive DBA was not needed, this thing runs itself. They then install the database, take the defaults and get the product up and going.

While this strategy may work on a dev or test system or even a smaller database, it fails to work/scale/manage well once you get into the larger space.

For example, a common situation as seen in Diagram 3, taking the default of one file for the transaction log and the data files. On the surface this may seem fine—install the database in one physical file, set it on autogrow and forget about it. What happens next is this... A year or two later, you run out of space on the drive. This again, doesn’t cause a problem—you just add a new file to a new drive letter and you are up and going. The problem happens when you find out that one drive is still getting hammered and the new drive you added is hardly being touched. This is because all the “old” data that you are constantly querying on is still in the original file, it is not balanced onto the new file. Only the new data added will go into the new file, so you still are left with a performance problem.

The wise thing to do is to create multiple files in the beginning that are filled with data equally, even if they are on the same physical disk drive in the beginning. What this allows you to do is move the files around later on to

alleviate the hot spot on a single disk drive. For example, if you have 4 files on one disk, you could `sp_detach_db`, `XCOPY` the files to a new location, then `sp_attach_db` and be up and going. By having multiple files from the very beginning, you have bought yourself a future management and performance path. For example, each new file could be on a different drive and different HBA channel and give a significant boost in performance. Having only one file limits your growth path. This is especially true with TEMPDB as discussed in KB Articles^v. Guidance in this area is to have one file per CPU core. For example, 8 cores = 8 database files. There are practical limits whereas it's more of a maintenance problem beyond 16. Additionally, you want all the files to be of uniform size to ensure the data is equally balanced across them all as it's being inserted. This is called proportional file fill.^{vi}

Another default that people take is "Autogrow by 10%". People tend to use "autogrow" as a management tool as opposed to a "fail safe" mechanism. Autogrow should only be used as a last resort. Bad things tend to happen when a database fills up. For example, when a file fills up its most likely when a lot of work is happening—middle of a payroll job perhaps or middle of the business day. When a file fills up, then it starts to expand with the autogrow. During the time in which the database is expanding, *it suspends operation*. The reason is that the database does not know if there is enough space left on the disk drive for it to expand to. Suspending operation in the middle of the business day is bad. Applications time out, users stop doing work, etc... Additionally, the database needs to allocate and initialize that space, which also causes heavy duty disk operations. If the underlying disk is being shared by other databases (ex. Two databases are on the same D: drive), then the other database is going to start getting "15 second" errors due to the heavy load caused by the expanding database.

If the database autogrow is set as a percent, the growth time and file fragmentation at the file system level gets skewed, also. Ten percent of a 10 megabyte database is going to be shorter in time than ten percent of a 100 gigabyte database.

Tools & Metrics

A full discussion of all the tools and how to run them is beyond the scope of this paper and will be addressed in future ones. The following is a brief outline of the tools that are available, some basic metrics to watch for and what they really mean to you.

Good, Busy, Bad

Reasonably people can debate the following numbers listed in Table 1, but they are the starting point from which we'll measure from.

Good	10ms or less
Reasonable	10ms to 20ms
Busy	20ms to 50ms
Bad	> than 50ms

Table 1: Disk Response Times

As noted above, an enterprise grade SCSI disk drive gets between 3ms to 8ms.

The goal is to have the following response times:

- SQL Server data files (*.mdf & *.ndf) should be greater than 20ms
- SQL Server log files (*.ldf) should not be greater than 10ms

Failure to have the log respond in 10ms risks the transaction log causing the SQL Server to stall because the log buffer cannot be flushed fast enough.

Is It The Database?

A common issue that the DBA is often fending off is “The database is the cause of this, not my disk.” This can often be easily figured out by running a basic test. Create a big file, for example a database file. Make it larger than the cache on your disk subsystem. XCOPY the file from a SAN drive to a local drive. Measure the throughput in MB/sec via perfmon and also measure how many seconds it took to run the XCOPY itself. At that point, you can run a BACKUP to NUL: and compare the same numbers.^{vii} If you see the same latency numbers in the XCOPY as you do in the BACKUP command, then you know that the problem exists independent of the database.

Measuring The Numbers

There are two ways to measure the response times: current and historical.

Current is the method in which most people look at problems. They fire up Windows Performance Monitor (PerfMon) and then start looking. While this is good, it fails to take into account what has happened over the course of time. For example, it will not address the question “What has my response time been since the instance was started?” etc...

The reason why this is an important distinction is that without good baseline or historical information you are just guessing as to what “slow” is today versus what was “slow” yesterday or last week.

To Look At What’s Happening Now

To see what the current disk response time:

1. Start up perfmon:
 - a. Start -> Run -> perfmon.exe
2. Remove all the other default events that are currently there. This just removes distractions.
3. Add the Average Disk Sec/* events found under LogicalDisk:



Diagram 4: Adding Disk Latency PerfMon Counters

Reading the PerfMon Data

The perfmon data will be listed in a decimal format such as “0.010”. This translates to 10ms (milliseconds). Per Table 1, our response times need to look like this:

Good	0.010 or less
Reasonable	0.010 to 0.020
Busy	0.020 to 0.050
Bad	> than 0.050

Table 2: Response Times in Decimal from PerfMon

Note: It's also important to point out that different vendors have proprietary perfmon counters. For example, the Veritas file system, EMC and HP have counters that monitor their disk response times. Please use them when needed.

What's Happened In The Past

There are two ways to look at historical data. One is to query the database itself which can give you the data since the instance has been started. The second is to setup a perfmon trace, sample every 15 seconds, and log it out to a file for analysis later on.

Querying the Database

Run the following query and it will show you how many milliseconds per IO it took for each file in the database. It's a good first pass when looking for IO issues to see where the hot spots are on your system:

```
select b.DbId, substring(b.name,1,10) 'Database Name', FileId,
IoStallMS/(NumberReads+NumberWrites) 'ms/io', filename
from ::fn_virtualfilestats(-1,-1) a, master..sysdatabases b
where a.DbId = b.dbid
```

DbId	Database Name	FileId	ms/io	filename
1	master	1	45	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\master.mdf
1	master	2	5	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\master.mdf
2	tempdb	1	105	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\tempdb.mdf
2	tempdb	2	30	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\tempdb.mdf
3	model	1	3	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\model.mdf
3	model	2	5	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\model.mdf
4	msdb	1	25	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\MSDBData.mdf
4	msdb	2	13	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\MSDBData.mdf

Diagram 5: SQL Script for looking at Historical Performance and it's output.

In the above, you can see that tempdb was getting his especially hard with 105ms disk access times.

One point to remember when looking at this information, it's an average since the server was up. Hence, if you see 105ms access times, this probably means it was doing nothing in the middle of the night and being slammed hard during the day—which means that the access times during the work day were significantly higher than 105ms.

To see how long the SQL Server has been up, check the errorlog.

PerfMon: Getting A Baseline

The great thing about perfmon is that it can be used in a reactive mode (something is broke, let's look at it now), and in a proactive mode (let's grab some information and see if we can spot trends over time).

To really know if something is "broke" or "slow" you need historical data to compare your current problem to. If someone says it's "slow" today, then can you look back on last Tuesday and see if the performance was the same or better? If not, you are just guessing. Use perfmon to collect this type of baseline data.

To collect a perfmon trace for historical information:

1. Start perfmon
2. Expand the "Performance Logs and Alerts"
3. Select "Counter Logs"
4. Right click on "Counter Logs"
5. Select "New Log Settings"
6. Give it a name
7. Select "Add Counters"
8. Select "LogicalDisk"
9. Select "Avg. Disk sec/[Read | Write | Transfer]"
10. Select all "All Instances"
11. Click "Add"
12. Click "Close"
13. Leave the interval at 15 seconds
14. Select the "Log Files" tab
15. Select "Text File (Comma delimited)" in the drop down
16. Click "Apply" and "Ok"

Now you will see your trace running in the counter logs and it should be "green" in color. To disable it, right click and select "Stop".

The output from this trace is stored in the C:\Perflogs directory.

The secret to using this data is Excel. Take the output CSV file in the Perflogs directory and import it into Excel. Then simply select the data range for the LUN you want to examine and click "Insert Line Charts" and Excel will graph the data out for you. You will see all the trends and patterns over the sampled time frame selected.

The following graph is output over 28 hours sampled every 15 seconds.

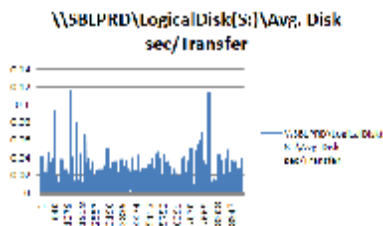


Diagram 6: Graphed Excel Data

What you see in the graph are disk times spiking over 100ms (.1) and averaging around 30ms to 40ms regularly (.04)—these are both red flags.

Best Advice

Run a perfmon trace and sample every 15 seconds and log the data. Collect the data over time. This will serve two purposes:

- You can see where the issues are. For example, when running payroll, our system bogs down. In addition to giving you firm data to work from in case someone complains that the system is “slow”. In that case, you can compare historical data to what it happening currently.
- The second benefit to collecting disk data, is you can also collect memory and CPU information, too. Once you graph this information, you can use it to help plan over time. If you see that you were using 23% CPU six months ago, 42% last month, and 56% this month—then you can plot and see that in 6 months from now, you will need a CPU upgrade. Hence, the data helps you do fiscal planning for the next budget cycle. It’s a crystal ball that helps you see into the future based on current and past growth patterns.

How To Fix It

Up until now, we’ve been talking about “what the issue is”, “how it occurs”, “how to monitor it”—now it’s time to address how to “fix it”.

As with most problems, you have these choices:

- Do nothing and let the problem continue.
- Throw hardware at the problem.
- Work smarter.

The answer is usually a combination of the last two.

What the Fix Usually Is: The Bartender Argument

You are at a party. There is plenty of beer, but the line is long. How do you fix the problem? Hire more bartenders.

Databases are exactly the same way. How do you increase the IO? Buy more disk and HBAs.

The issue often is not space—you got plenty of disk (beer in the above analogy). The issue is speed—you need more physical disk arms going after the data to service the requests (more bartenders).

Generally speaking, the fix is to add more physical spindles and HBAs to the IO subsystem. It’s typically as easy as that.

How Much Hardware Do I Need?

This is the magic question that everyone wants to know. This is really a sizing exercise that is provided by hardware vendors. There are guides out there on this subject and sizing spreadsheets. While this is outside the scope of this document, the author will be writing a paper on this topic in the future.

The key argument to press with your hardware vendor is this: Given my current workload, I need 10ms disk response times. This keeps you above the fray and from having to really dig into IOPS calculations, RAID configurations, etc...

What we can do in this paper are show you the tools to use to create a workload to see if the disk can handle the hardware that you may purchase.

Tools for Creating a Load

You can record a profiler trace and play it back using a number of different tools that are freely available.^{viii} This allows you to play back a real live workload on another set of hardware.

Additionally, there are many tools available that can create a simulated workload on a disk subsystem and report back the response times and throughput. The following is a brief list of them:

- SQLIO^{ix}
- SQLIOSIM^x
- IOMETER^{xi}

The main difference between SQLIO and SQLIOSIM is this:

SQLIO creates a workload to test IO performance and throughput. SQLIOSIM is a tool that checks the reliability of a storage subsystem, although it does imply a given level of load.

IOMETER is good because it's OS independent. Chances are your SAN guys know about it already. You can use the same tool on Unix, Linux, Windows, etc... The down side is that it doesn't create SQL Server based patterns. Please see the SQL IO paper listed in the references section to understand the types of IO SQL Server generates so you can have IOMETER make similar patterns.

A key consideration that is often overlooked when testing is the length of time the test should be run. The reason this is pointed out is twofold:

- Cache management technology on a SAN is adaptive. The longer a test is run, the better it tunes itself. Consult with your SAN provider for information on this. At least run the test for an hour.
- If something is going to break, it tends to do so in the first month. If possible, consider running patterns and burning the SAN in over the course of a month.

Considerations for Architecting the Disk Subsystem

Note: The following discussion is more for about guidance. Consult with a disk hardware specialist for specifics to your system.

RAID

RAID technology enables resiliency and performance for a disk subsystem. Often time's people don't understand the implications of implementing certain types of solutions. For example, no one ever got fired for saying, "Go with RAID 0+1". Why? Because it provides for the most protection, but it may not be the best for performance because it will require you to buy twice as many spindles as you need. Conversely, people say, "Go with RAID 5" because it provides a balance between economy and resiliency. The problem with that argument is that RAID 5 seriously impairs performance on write intensive parts of the database such as the transaction log and tempdb.

As you can tell, there is no "easy" way through this discussion. The answer usually comes from the most basic questions: What is your SLA? And, What is your budget?

Most OLTP systems do 95% to 98% reads (SELECT) and 2% to 5% writes (INSERT/UPDATE/DELETE). This can be verified by running a profiler trace and analyzing the IO patterns. For example, counting the SELECT's or as easy as just running a SUM(Reads) /SUM(Writes) on the data captured and looking at the ratio between the two values.

Different RAID levels offer different types of protection and there is an associated cost with each one. The following table illustrates the overhead for calculating the parity bit associated with each type:

RAID Level	
1, 0+1, 10	20%
2	10%
3, 30	25%
4	33%
5, 50	43%

Table 3: RAID Level and Parity Bit Calculation Penalty

For example, in a RAID 0+1 configuration (mirroring and striping), the data has to be written to two different locations and it will take 20% more time than if you were just writing to a RAID 0 stripe with no mirroring.

Additionally, different RAID levels consume more disk than others to protect from failure and data loss. The following table illustrates this:

	Amount of Usable Disk	Comments
RAID 0	All disk used	High performance, no protection.
RAID 1	50% of disk used	No striping for performance.
RAID 5	$(\text{Number of Drives} - 1) / (\text{Number of Drives})$	Good protection, good read performance, fair write performance.
RAID 6	$(\text{Number of Drives} - 2) / (\text{Number of Drives})$	Typically used with less reliable and inexpensive SATA drives
RAID 0+1	50% of disk used	Maximum protection and high performance.

Table 4: RAID Level and Amount of Usable Disk

For example, if you have 20 disk drives in RAID 0+1 you get effective use of 10 disk drives. This is a reduction of 50% of your resources. Whereas, if you have 20 disk drives in RAID 5 you get effective use of 19 disk drives based on the formula above of $(\text{Number of Drives} - 1) / (\text{Number of Drives}) = (20-1)/(20) = .95 = (20 * .95) = 19$ disk drives. RAID 5 gives 90% more spindles than RAID 0+1 in a typical OLTP database (per this example).

For a fuller discussion of how to layout your disk please consult the End Notes section of this paper which provides other sources to visit for direction in this area.^{xii}

The Biggest Mistakes

The biggest mistake commonly seen is one of the two scenarios:

1. The customer puts everything on RAID 5—data, transaction log, and tempdb. Sometimes they put them on separate RAID 5 LUNs, but often they put them on the same LUN. Either situation performs poorly due to the 43% overhead on the write statements on the tempdb or log. The time used for calculating this penalty causes the latency to exceed 10ms easily.
2. The customer puts everything on RAID 0+1. On the surface, this looks great but when they kick up a real workload on the system, they find that using only 50% of the disk chokes off the IO subsystem and they have high latency times.

Best Economical Solution

Success in IO is measured by how many disk and how many IO channels you have under you. In an OLTP system where most work are SELECTS, it makes sense to put the data on a RAID 5 LUN. Using RAID 5 gives you parity protection for loss of a physical disk in the array, and also speed by putting more physical disk under you than a RAID 0+1 solution. TempDB is usually about 10% the size of the database and so is the transaction log. Hence, you can afford to put these on 0+1 devices.

Throughput

Simply put, throughput is the measure of how many megabytes per second you can push from the disk subsystem to the database. It is independent of disk latency times. For example, my disk can be very fast, but if I don't have an IO channel big enough to push the data through, then I'm going to bottleneck on throughput. Operations that are dependent on throughput are backups and large scans, and applications such as a data warehouse.

Throughput is also dependent on transfer size. When comparing a large transfer size versus a small transfer size on the same amount of disk and HBAs, large transfer sizes will always push through more MB/sec. For example, 1000 IOs per second (IOPS) on an OLTP system doing 8K reads versus 1000 IOPS on an OLAP system doing 1MB scans the MB/sec would roughly be 9MB/sec for the OLTP and 230MB/sec for the OLAP.^{xiii} In the OLAP system, you saturate the bus with data much faster than small IO on the OLTP. Smaller transfer sizes still need high IOPS capacity subsystem, but do not generate high MB/sec requirements.

Don't go by what the HBA vendor posts on their spec sheet for throughput. These numbers are often made when they are disk to disk with no operating system in between. The following table will help provide guidance:

	Usable	MB/sec
Fiber SCSI	80%	256
Copper SCSI	70%	224
IDE	60%	192
2GB HBA		150
1GB HBA		80

Table 5: Usable Bandwidth of HBA

Finally, the more physical disk and the faster they are (10K vs. 15K) you have under you, the higher the IOPS will be and the higher the MB/sec can be for the same workload.

Basic Rule: The way to increase throughput is to stripe the data over as many physical disk and HBAs as possible. This will get multiple disk armatures and multiple IO channels working in parallel as opposed to sequential.

The Fear: Low throughput can have unintended consequences. For example, any type of operation that in maintenance oriented will run long: rebuilding indexes, updating statistics and backups. A long running backup is a sign that the throughput is not meeting requirements.

Best Advice: Buy more bandwidth. HBAs are cheap. They provide higher throughput and also more resiliency in case of failure.

Case Study: Starved On Spindles & IO Channels

A customer was having performance issues on not getting proper throughput. It was recommended they buy a second HP MSA60 to double the amount of physical disk they had and get another IO channel going to the disk. The following were the results:

Issues:

- One big RAID device (0+1).
- One big file.
- Everything on one drive.
- Not enough drives or channels.
- Starved on disk.
- Mixed SCSI speeds.

Fix:

- Put on another DAS and use two channels.
- Use RAID 5 for data and 0+1 for tempdb and log.
- Give more disk for data.
- Use multiple data files.

Cost: \$6,556

Baseline from 10/27/06:

BACKUP DATABASE successfully processed 8259522 pages in 638.561 seconds (105.960 MB/sec).

After database work 3/4/07:

BACKUP DATABASE successfully processed 9497102 pages in 313.501 seconds (248.165 MB/sec).

Results: 134% increase in throughput performance.

Case Example: Throughput With Different RAID Levels

This example shows the relationship between the number of disk in a LUN and the amount of throughput that can be achieved by adding more disk to the stripe. The following backup test was run on an HP MSA 60, 4 x 72GB 15K SAS, P800 SmartArray with 512MB cache.

RAID 0: No Parity. All disk used.

BACKUP DATABASE successfully processed 15018185 pages in 355.591 seconds (345.984 MB/sec).

RAID 5: One parity disk. Only 3 disk used.

BACKUP DATABASE successfully processed 10172177 pages in 295.428 seconds (282.066 MB/sec).

RAID 6: Two parity disks. Only 2 disk used.

BACKUP DATABASE successfully processed 10059329 pages in 395.086 seconds (208.577 MB/sec).

What you see is that by using more physical disk the throughput goes up significantly. Please note the relationship between throughput and IOPS here. This LUN has only four disk, yet gets throughput better than a lot of very expensive SANs—but it does not have enough physical disk to handle a large workload in a busy OLTP system.

The Space Dilemma and The SAN Fight

Arguably the biggest point of contention when working with customers and sizing comes down to this: They end up buying a terabyte of disk space to handle a 100GB database. The reason is that they can't buy small disk drives. As illustrated in the examples above, the secret to good disk performance are the number of spindles NOT the amount of space.

The result goes something like this... in a few months (or sooner), someone else needs space for their application. They see the LUN that your database sits on has plenty of free bytes left. They install their app on your physical disk. Next you start experiencing very poor performance, yet your application has not changed nor has the database. Upon further inspection, you find the that the offending app is the one that has been co-mingled with yours.

The only way to get consistent performance is for you to have dedicated spindles/resources for your application.

This presents a huge problem to your SAN/infrastructure team. They sold management on this idea of a centralized utility that will save them money by letting them consolidate all their storage into one location. Better security, backing up data, etc... And when you tell them that they cannot share your space and that you need dedicated physical drives—you have just ruined their TCO story. All of a sudden the cost savings that they built went out the door. If they got 1TB of space, they want to use it. SAN guys tend to look at disk in terms of space, not performance. They give you RAID 5 chunks without any consideration as to the app on top of it. As discussed before, these are all ideas around "Poor Performance by Design". In the end, all you can say is "Give me 10ms disk access times for my workload." Just be aware of this discussion and how it goes.

Work Smart: DTA + DMVs

As suggested earlier, if a disk is getting hammered, there are two ways to solve the problem: add more hardware or lighten the load on the disk. This section will briefly discuss two methods for lightening the load.

SQL Server 2005 provides a new set of tools called Dynamic Management Views (DMVs). These tools allow the DBA to see more about what's going on inside the SQL Server instance. Specifically, SQL Server 2005 now tells the DBA what indexes are missing and how to add them. The following script queries the DMVs and builds the missing indexes SQL Server would like. Please note, this script is NOT a substitute for careful analysis!

```
SELECT
    mig.index_group_handle, mid.index_handle,
    CONVERT (decimal (28,1),
        migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user_scans)
    ) AS improvement_measure,
    'CREATE INDEX missing_index_' + CONVERT (varchar, mig.index_group_handle) + '_' + CONVERT (varchar, mid.index_handle)
    + ' ON ' + mid.statement
    + ' ( ' + ISNULL (mid.equality_columns,'')
    + CASE WHEN mid.equality_columns IS NOT NULL AND mid.inequality_columns IS NOT NULL THEN ',' ELSE '' END
    + ISNULL (mid.inequality_columns, '' )
    + ' )'
    + ISNULL (' INCLUDE ( ' + mid.included_columns + ' )', '') AS create_index_statement,
    migs.*, mid.database_id, mid.[object_id]
FROM sys.dm_db_missing_index_groups mig
INNER JOIN sys.dm_db_missing_index_group_stats migs ON migs.group_handle = mig.index_group_handle
INNER JOIN sys.dm_db_missing_index_details mid ON mig.index_handle = mid.index_handle
WHERE CONVERT (decimal (28,1), migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user_scans)) > 10
ORDER BY migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user_scans) DESC
```

Diagram 7: Build Missing Indexes Script^{xiv}

By building better indexes, hopefully you can lighten up the load on the disk and the resources can be better shared across the rest of the database server, hence lessening the chances of a time out.

Another tool to use is SQL Server 2005's Database Tuning Advisor (DTA). DTA is a vastly updated version of SQL Server 2000's Index Tuning Wizard. The good news is that *DTA is backwards compatible with SQL Server 2000*. If you have a problem with an application in 2000, simply point the DTA from 2005 back to the 2000 database and let it resolve the issue.

DTA performs exhaustive analysis that suggest indexes, statistics, materialized views, and partitioning that DMVs do not.

A full discussion of this tool is beyond the scope of this paper, but please be aware that DTA can significantly lighten the workload by creating indexes for you.

Summary

The goal of this paper was to expand out what appears to be an informational message in the error log and explain not only what it's meaning are, but suggest different why it happens and ways to really solve this problem.

In the past, disk performance problems were not so much an issue. The reason was that physical disk were small which forced people to buy more of them. By having many small disk, they didn't see as many IO bottlenecks. But today, as the disk keep getting larger, but the speeds remain the same—we are seeing more and more disk bottleneck issues with SQL Server because there are fewer spindles to support the workload.

Hopefully this paper answered some of the questions you had and gave some direction for proactively creating a better solution.

References

Microsoft Knowledge Base Article: 897284

SQL Server 2000 SP4 diagnostics help detect stalled and stuck I/O operations

<http://support.microsoft.com/kb/897284>

Maxtor Drive Specifications

Disk drive speeds average 3ms to maximum 8ms.

http://www.maxtor.com/files/maxtor/en_us/documentation/data_sheets/atlas_15kii_data_sheet.pdf

SQL IO Basics

Very detailed paper on how SQL Server does IO.

<http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspix>

SQL Server Disk Response Times

This paper talks about SQL Server pre-deployment testing for disk drives, tools used, and acceptable response times. The following link points directly to what acceptable response times are:

<http://www.microsoft.com/technet/prodtechnol/sql/bestpractice/pdpliobp.mspix#EYIAC>

Physical Database Storage Design

The following paper offers an overview of physical database storage design.

<http://www.microsoft.com/technet/prodtechnol/sql/2005/physdbstor.mspix>

RAID

The following sites offer a brief overview as to what RAID is and what the levels mean.

<http://msdn2.microsoft.com/en-us/library/ms190764.aspx>

<http://en.wikipedia.org/wiki/RAID>

TempDB Guidance

The following articles give guidance on sizing of tempdb.

<http://support.microsoft.com/default.aspx?kbid=834846>

<http://support.microsoft.com/default.aspx?scid=kb;en-us;328551>

<http://www.microsoft.com/technet/prodtechnol/sql/2005/workingwithtempdb.mspix>

Annotated List of Blog Entries Referenced In This Paper

SQL Server Urban Legends Discussed

Bob Dorr in follow up comments on June 6, 2007 talks about multiple data files:

<http://blogs.msdn.com/psssql/archive/2007/02/21/sql-server-urban-legends-discussed.aspx>

Proportional File Fill

Juergen Thomas gives some great tools on how to look at data distribution and how proportional file fill works:

<http://blogs.msdn.com/saponsqlserver/archive/2007/12/01/proportional-fill-and-how-to-check-whether-it-works.aspx>

How to Run A BACKUP to NUL:

In the Database+Disk+Performance Blog, an entry illustrates how to run the test, what it does, it's uses and how to read the results:

http://databasediskperf.blogspot.com/2006/06/disk-performance-and-backup-to-nul_20.html

Create Missing Indexes Based on DMV Results

This blog entry by Bart Duncan gives you a simple script to run the builds indexes for you based on what SQL Server thinks it needs based on the current work load. Very useful and clever:

<http://blogs.msdn.com/bartd/archive/2007/07/19/are-you-using-sql-s-missing-index-dmvs.aspx>

Vocabulary

HBA. Host Bus Adapter. This is the controller in which the disk drives are connected up to the server. This may be fiber or copper.

Latency. In the context of this paper, latency is the response time as is measured by the OS for the disk to respond.

LUN. A LUN is a logical drive (letter) that represents a collection of physical disk drives.

RAID. A collection of physical disk in an array that provides some type of striping (for performance) and/or parity enablement (for redundant protection).

ROI. A business term for "Return on Investment".

Spindle. A spindle is synonymous in this discussion with a physical drive.

TCO. Total Cost of Ownership is a business term that refers to how much it cost to run a solution end-to-end. For example, operations, hardware, software, and ongoing maintenance.

Credits

Bob Dorr
Bart Duncan
Keith Elmore
Wanda He
Peter Kalbach
Juergen Thomas
Steven Wort

End Notes

- ⁱ As an example, see the Maxtor spec sheet on the Atlas II drive.
- ⁱⁱ See the paper listed in the references section on SQL Server Disk Response Times.
- ⁱⁱⁱ See SQL Server IO paper listed in the References section for how the Write Ahead Log (WAL) protocol works.
- ^{iv} See KB article 897284 for a listing of these issues: <http://support.microsoft.com/kb/897284/>
- ^v See KB article <http://support.microsoft.com/default.aspx?scid=kb;en-us;328551> for tempdb guidance.
- ^{vi} Read the blog post by Juergen Thomas of SQL Server for how proportional file fill works: <http://blogs.msdn.com/saponsqlserver/archive/2007/12/01/proportional-fill-and-how-to-check-whether-it-works.aspx>
- ^{vii} Please read the blog posting “backing up to a NUL: device”:
http://databasediskperf.blogspot.com/2006/06/disk-performance-and-backup-to-nul_20.html
- ^{viii} You can use OSTRESS and Profiler to play back traces. This will let you play back real live workloads to simulate how much activity is taking place. To get OSTRESS:
<http://support.microsoft.com/kb/887057>
- ^{ix} SQLIO is located at:
<http://download.microsoft.com/download/f/3/f/f3f92f8b-b24e-4c2e-9e86-d66df1f6f83b/SQLIO.msi>
- ^x SQLIOSIM is located at:
<http://support.microsoft.com/kb/q231619/>
- ^{xi} IOMETER is located at:
<http://www.iometer.org/doc/downloads.html>
- ^{xii} Please read “TechNote 9” on the Microsoft-Oracle website. This note draws a matrix for starting points in laying out your database based on size of database and size of budget. Though this note is written for Siebel, it’s useful for most ERP and CRM systems:
<http://www.microsoft-oracle.com/Siebel/Pages/TechnicalNotes.aspx>
- ^{xiii} Assumptions: 32 15K rpm SCSI on 4 HBAs for the OLAP, 32 15 rpm SCSI on 1 HBA for the OLTP.
- ^{xiv} This index creation script is from Bart Duncan’s blog:
<http://blogs.msdn.com/bartd/archive/2007/07/19/are-you-using-sql-s-missing-index-dmvs.aspx>