Getting the hang of IOPS

If you are an Altiris Administrator, take it from me that IOPS are important to you. What I hope to do in today's article is to help you understand what IOPS are and why they are important when sizing your disk subsystems. In brief I cover the following,

- Harddisk basics -how harddisks work!
- Drive response times
- Interpreting drive throughputs -what these figures actually mean
- What IOPS are and why they are so important
- IOPS calculations and disk arrays

I should state now that I do not consider myself an expert on this topic. However, every so often I find myself benchmarking disks, and I know the curve I had to climb to interpret all the various vendor stats -the information overload can be overwhelming. What I'm going to attempt in this article is to herd together all the salient pieces of information I've gathered over time. With luck, this will help you engage in a meaninful dialog with your storage people to get the performance you need from your storage.

Introduction

If you are looking at IT Management Suite (ITMS) one of the underpinning technologies which needs to be considered in earnest is Microsoft SQL Server. Specifically, you want to be sure that your SQL server is up to the job. There are many ways to help SQL Server perform well. Among them are,

- Move both the server OS and the SQL Server application to 64-bit
- Ensure you've got enough RAM chips to load your entire SQL database into memory
- Ensure you've got enough processing power on-box
- Ensure the disk subsystem is up to the task
- Implement database maintenance plans
- Performance monitoring

One of the most difficult of the line items to get right in the above list is ensuring the disk susbsystem is up to the task. This is important -you want to be sure that the hardware you are considering is suitable *from the outset* for the loads you anticipated placing on your SQL Server.

Once your hardware is purchased, you can of course tweak how SQL server utilises the disks it's been given. For example, to reduce contention we can employ different spindles for the OS, databases and log files. You might even re-align your disk partitions and tune your volume blocksizes when formatting.

But specifying the disk subsystem initially leads to a lot of tricky questions,

1. How fast *really* are these disks?

2. Okay I now know how fast they are. Err... Is that good?
3. Is the disk configuration suitable for SQL requirements of ITSM 7.1?

Before we can begin to answer these questions, we really need to start at the beginning...

Disk Performance Basics

Disk performance is an interesting topic. Most of us tend to think of this in terms of how many MegaBytes per second (MB/s) we can get out of our storage. Our day-to-day tasks like computer imaging and copying files between disks teaches us that this MB/s figure is indeed an important benchmark.

It is however vital to understand that these processes belong to a specific class of I/O which we call *sequential*. For example, when we are reading a file from beginning to end in one continuous stream we are actually executing a *sequential read*. Likewise, when copying large files the write process to the new drive is called a *sequential write*.

When we talk about rating a disk subsystem's performance, the sequential read and write operations are only half the story. To see why, let's take a look into the innards of a classic mechanical harddisk.

Hard Disk Speeds - It's more than just RPM...

A harddisk essentially consists of some drive electronics, a spinning platter and a number of read/write heads which can be swung across the disk on an arm. Below I illustrate in gorgeous powerpoint art the essential components of a disk drive. Note I am focusing on the mechanical aspects of the drive as it is these which limit the rate at which we can read data from (and write data to) the drive.

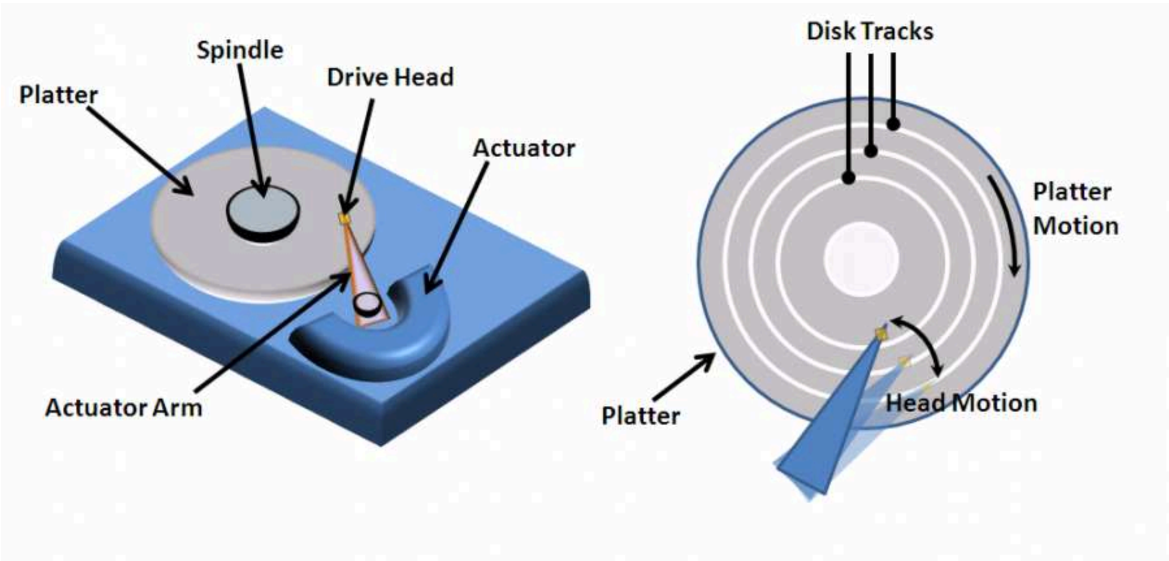


Illustration depicting disk (left) and the motion of the head across the tracks (right)

The main items in the above figure are,

1. *The Disk Platter*
   The platter is the disk within the drive housing upon which our information is recorded. The platter is a hard material (i.e. not floppy!) which is usually either aluminium, glass or a ceramic. This is coated with a magnetic surface to enable the storage of magnetic bits which represent our data. The platter is spun at incredible speeds by the central spindle (up to 250kmph on the fastest disks) which has the effect of presenting a stream of data under the disk head at terrific speeds.

   In order to provide a means to locate data on the disk, these platters are fomatted with thousands of concentric circles called *tracks*. Each track is subdivided into sectors which each store 512 bytes of data.

As there is a limit to the density with which vendors can record magnetic information on a platter, manufacturers will often be forced to make disk drives with several platters in order to meet the storage capacities their customers demand.

2. *The Drive Head*
   This is the business end of the drive. The heads read and write information bits to and from the magnetic domains that pass beneath it on the platter surface. There are usually two heads per platter which are sited on either side of the disk.
3. The Actuator Arm
   This is the assembly which holds the heads and ensures (through the actuator) that the heads are positioned over the correct disk track.

When considering disk performance one of the obvious players is the platter spin speed. The drive head will pick up far more data per second from a platter which spins at 1000 **R**otations **P**er **M**inute (RPM) when compared with one that spins just once per minute! Simply put, the faster the drive spins the more sectors the head can read in any given time period.

Next, the speed with which the arm can be moved between the disk tracks will also come into play. For example, consider the case where the head is hovering over say track 33 of a platter. An I/O request then comes in for some data on track 500. The arm then has to swing the head across 467 tracks in order to reach the track with the requested data. The time it takes for the arm to move that distance will fundamentally limit the number of random I/O requests which can be serviced in any given time. For the purposes of benchmarking, these two mechanical speeds which limit disk I/O are provided in the manufacturer's specification sheets as *times,*

1. *Average Latency*
   This is the time taken for the platter to undergo half a disk rotation. Why half? Well at any one time the data can be either a full disk rotation away from the head, or by luck it might already be right underneath it. The time taken for a half rotation therefore gives us the *average* time it takes for the platter to spin round enough for the data to be retrieved.
2. *Average Seek Time*
   Generally speaking, when the I/O request comes in for a particular piece of data, the head will not be above the correct track on the disk. The arm will need to move so that the head is directed over the correct track where it must then wait for the platter spin to present the target data beneath it. As the data could potentially be anywhere on the platter, the average seek time is time taken for the head to travel half way across the disk.

So, whilst disk RPM is important (as this yeilds the average latency above) it is only half the story. The seek time also has an important part to play.

The Response Time

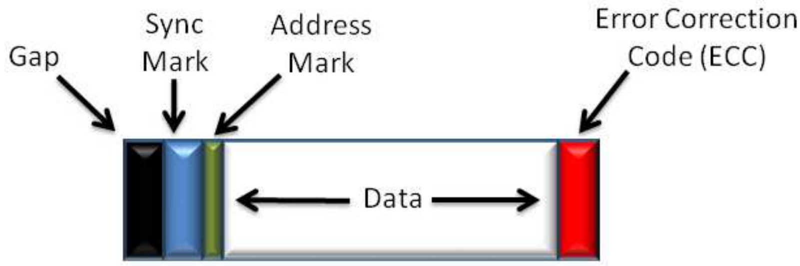
Generally speaking, the time taken to service an individual (and random) I/O request will be limited by the combination of the above defined latency and seek times. Let's take for example a fairly mainstream retail laptop harddisk -a Seagate Momentus. From the Seagate website its specifications are,

Spin Speed (RPM) .... 7200 RPM
Average latency 4.17ms
Seek time (Read) 11ms
Seek time (Write) 13ms
I/O data transfer rate 300MB/s

Returning to our special case of a sequential read, we can see that the time taken to locate the start of our data will be the *sum* of the *average latency* and the *average seek* times. This is because once the head has moved over the disk to the correct track (the seek time) it will still have to wait (on average) for half a platter rotation to locate the data. The total time taken to locate and read the data is called the drive's *response time*,

Response Time = (*Average Latency)* + (*Average Seek Time)*

I've heard people question this formula on the grounds that these two mechanical motions occur concurrently -the platter is in motion whilst the arm is tracking across the disk. The thinking then is that the response time is which ever is the larger of seek and latency. This thought experiment however has a flaw -once the drive head reaches the correct track it has no idea what sector is beneath it. The head only starts reading once it reaches the target track and thereafter must use the sector address marks to orient itself (see figure below). Once it has the address mark, it knows where it is on the platter and therefore how many sector gaps must pass before the target sector arrives.



Graphical Illustration of Sector Layout on a Hard disk

The result is that when the head arrives at the correct track, we will still have wait on average for half a disk rotation for the correct sector to be presented. The formula which sums the seek and latency to provide the drive's response time is therefore correct.

Digression aside, the response time for our Seagate Momentus is therefore,

```
(Response Time) = 11ms + 4.17ms

                = 15.17ms.
```

So the drive's response time is a little over 15 *thousandths* of a second. Well that sounds small, but how does this compare with other drives and in what scenarios will the drive's response time matter to us?

To get an idea of how a drive's response time impacts on disk performance, let's first see how this comes into play in a sequential read operation.

Disk Transfer Rates aka the 'Sequential Read'

Most disk drive manufacturers report both the response time, and a peak transfer rate in their drive specification. The peak transfer rate typically refers to the best case sequential read scenario.

Let's assume the OS has directed the disk to perform a large sequential read operation. After the initial *average* overhead of 15.17ms to locate the start of the data, the actuator arm need now move only fractionally with each disk rotation to continue the read (assuming the data is contigious). The rate at which we can read data off the disk is now limited by the platter RPM and how much data the manufacturer can pack into each track.

Well, we know the RPM speed of the platter, but what about the data density on the platter? For that we have to dig into the manufacturers spec sheet,

Drive specification	ST95005620AS	ST93205620AS	ST92505610AS
Formatted GB (512 bytes/sector)*	500	320	250
Guaranteed sectors	976,773,168	625,142,448	488,397,168
Bytes per sector	512		
Physical read/write heads	4	3	2
Discs	2		1
Cache (MB)	32		
Recording density in BPI (bits/in max)	1,490k		
Track density TPI (tracks/in max)	265k		
Areal density (Gb/in^2 max)	394		
Spindle speed (RPM)	7200		
Average latency (ms)	4.17		
Internal transfer rate (Gb/s max)	1.23		
I/O data transfer rate (Gb/s max)	3.0		

This tells us that the number of bits per inch of track is 1,490,000. Let's now use this data to work out how much data the drive could potentially deliver on a sequential read.

Noting this is a 2.5inch drive, the maximum track length is going to be the outer circumference of the drive (pi * d) = 2.5*3.14 = 7.87 inches. As we have 1490kb per inch data density, this means the maximum amount of data which can be crammed onto a track is about,

```
(Data Per Track)  = 7.87 * 1490 k bits

                 = 11,734 k bits

                 = 1.43MB
```

Now a disk spinning at 7200RPM is actually spinning 120 times per second. Which means that the total amount of data which can pass under the head in 1 second is a massive 173MB (120 * 1.43MB).

Taking into account that perhaps about 87% of a track is *data*, this gives a maximum disk throughput of about 150MB/s which is surprisingly in agreement with Seagates own figures.

Note that this calculation is *best case* -it assumes the data is being sequentially read from the outermost tracks of the disk and that there are no other delays between the head reading the data and the operating system which requested it. As we start populating the drive with data, the tracks get smaller and smaller as we work inwards (don't worry -we'll cover this in Zone Bit Recording below). This means less data per track as you work towards the centre of the platter, and therefore the less data passing under the head in any given time frame.

To see how bad the sequential read rate can get, let's perform the same calculation for the smallest track which has a 1 inch diameter. This gives a worst case sequential read rate of 60MB/s! So when your users report that their computers get progressively slower with time, they might not actually be imagining it. As the disk fills up, retrieving the data from the end of a 2.5inch drive will be 2.5 times slower than retrieving it from the start. For a 3.5 inch desktop harddisk the difference is 3.5 times.

The degradation which comes into play as a disk fills up aside, the conclusion to take away from this section is that a drive's response time does not impact on the sequential read performance. In this scenario, the drives data density and RPM are the important figures to consider.

Before we move onto a scenario where the response time is important, let's look at how drives manage to store more data on their outer tracks than they do on their inner ones.

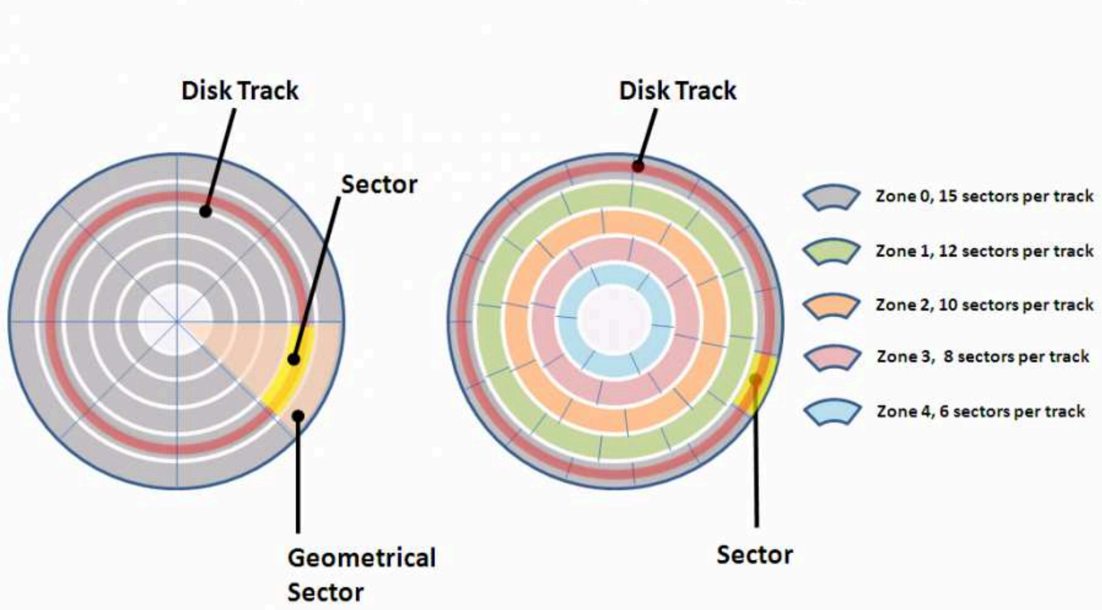
Zone Bit Recording

As I stated in the above section, the longer outer tracks contain more data than the shorter inner tracks. This might seem obvious, but this has not always been the case. When harddisks were first brought to market their disk controllers were rather limited. This resulted in a very simple and geometric logic in the way tracks were divided into sectors as shown below. Specifically, each track was divided into a *fixed number* of sectors over which the data could be recorded. On these disks the number of sectors-per-track was a constant quantity across the platter.

As controllers became more advanced, manufacturers realised that they were finally able to increase the complexity of the platter surface. In particular, they were able to increase the numbers of sectors per track as the track radius increased.

The optimum situation would have been to record on each track *as many sectors as possible* into its length, but as disks have *thousands* of tracks this presented a problem - the controller would have to keep a table of all the tracks with their sector counts so it would know *exactly* what track to move the head to when when reading a particular sector. There is also a law of diminishing returns at play if you continue to attempt to fit the maximum number of sectors into each and every track.

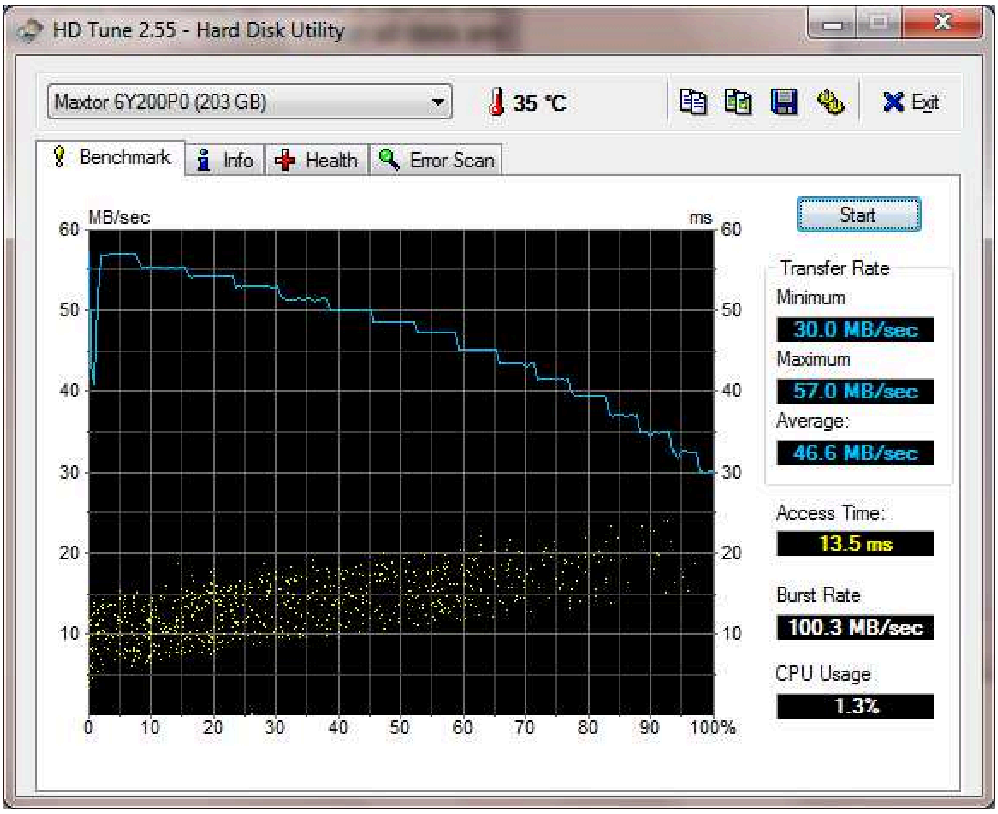
A compromise was found. The platter would be divided into a small number of *zones.* Each zone being a logical grouping of tracks which had a specific sector-per-track count. This had the advantage of increasing disk capacities by using the outer tracks more effectively. Importantly, this was achieved without introducing a complex lookup mechanism on the controller when it had to figure out where a particular sector was located.



Legacy disk sector layout with fixed sectors per track (left). Zone Bit Recording sector layout with variable sectors per track (right)

The diagram above shows an example where the platter surface is divided into 5 zones. Each of these zones contains a large number of tracks (typically thousands), although this is not illustrated in the above pictures for simplicity. This technique is called **Z**one **B**it **R**ecording, or ZBR for short.
On some harddisks, you can see this zoning manifest very clearly if you use a disk benchmarking tool like HD Tune. This tool tests the disk's sequential read speed working from the outermost track inwards. In the particular case of one of my Maxtor drives, you can see quite clearly that the highest disk transfer rates are obtained on the outer tracks. As the tool moves inwards, we see a sequence of steps as the read head crosses zones possessing a reduced number of sectors per track. In this case we can see that the platter has been divided into 16 zones.

.



This elegant manifestation of ZBR is sadly hard to find on modern drives -the stairs are generally replaced by a spiky mess. My guess is that other trickery is at play with caches and controller logic which results in so many data bursts as to obscure the ZBR layout.

Understanding Enterprise Disk Performance

Now that we've covered the basics of how harddisks work, we're now ready to take a deeper look into disk performance in the enterprise. As we'll see, this means thinking about disk performance in terms of response times instead of the sustained disk throughputs we've considered up to now.

Disk Operations per Second - IOPS

What we have seen in the above sections is that the disk's *response time* has very little to do with a harddisk's transfer rate. The transfer rate is in fact dominated by the drive's RPM and linear recording density (the maximum number of sectors-per-track)

This begs the question of exactly *when does the response time become important?*

To answer this, let's return to where this article started -SQL Servers. The problem with databases is that database I/O is *unlikely to be sequential in nature*. One query could ask for some data at the top of a table, and the next query could request data from 100,000 rows down. In fact, consecutive queries might even be for different databases.
If we were to look at the disk level whilst such queries are in action, what we'd see is the head zipping back and forth like mad -apparently moving at random as it tries ro read and write data in response to the incoming I/O requests.
In the database scenario, the time it takes for each small I/O request to be serviced is dominated by the time it takes the disk heads to travel to the target location and pick up the data. That is to say, the disk's *reponse time* will now dominate our performance. The response time now reflects the time our storage takes to service an I/O request when the request is random and small. If we turn this new benchmark on its head, we can invert this to give the number of **I**nput/**O**utput o**P**erations per **S**econd (**IOPS**) our storage provides.

So, for the specific case of our Seagate Drive with a 15.17ms response time, it will take at least *on average* 15.17ms to service each I/O. Turning this on it's head to give us our IOPS yeilds (1/ 0.01517) which is 66 IOPS.

Before we take a look and see whether this value is good or bad, I must emphasise that this calculation *has not taken into account the process of reading or writing data*. An IOPS value calculated in these terms is actually referring to zero-byte file transfers. As ludicrous as this might seem, it does give a good starting point for estimating how many read and write IOPS your storage will deliver as the response time will dominate for small I/O requests.

In order to gauge whether my Seagate Momentus IOPS figure of 66 is any good or not, it would be useful to have a feeling for the IOPS values that different classes of storage provide. Below is an enhancement to a table inspired by Nick Anderson's efforts where he grouped various drive types by their RPM and then inverted their response times to give their zero-byte read IOPS,

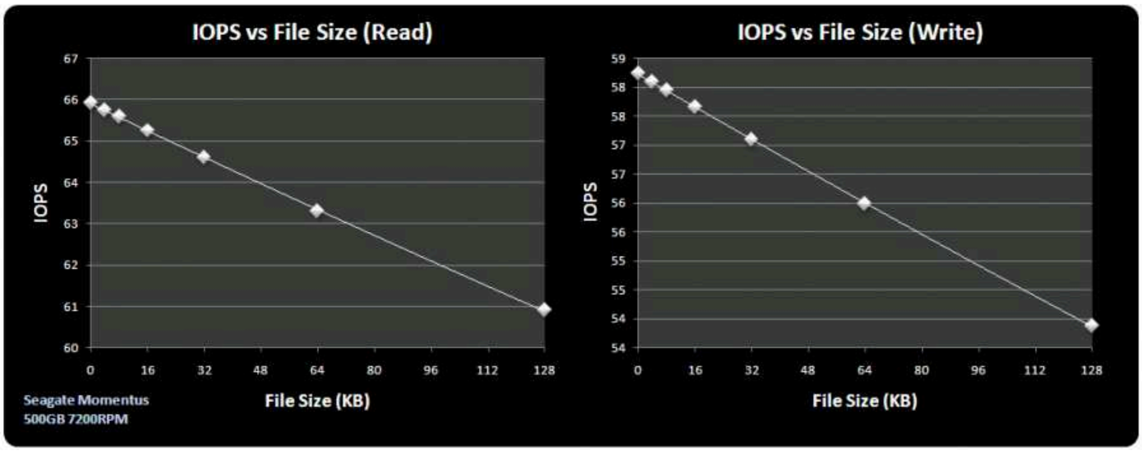
Drive RPM	Average Rotational Latency (ms)	Seek Time (ms)	IOPS
5400	5.5	7 - 14.4	50-80
7200	4.2	5.9 - 9.1	75-100
10,000	3	3.6 – 5	125-150
15,000	2	2.7 – 3.7	175-210

As you can see, my Seagate Momentus actually sits in the 5400RPM bracket even though it's a 7200RPM drive. Not so surprising as this is actually a laptop drive, and compromises are often made in order to make such mobile devices quieter. In short -your milage will vary.

IOPS and Data

Our current definition of a drive's IOPS is based on the time it takes a drive to retrieve a zero-sized file. Of immediate concern is what happens to our IOPS values as soon as we want to start retrieving/writing data. In this case, we'll see that both the response time and sequential transfer rates comes into play.

To estimate the I/O request time, we need to sum the response time with the time required to read/write our data (noting that a write seek is normally a couple of ms longer than a read seek to give the head more time to settle). The chart below therefore shows how I'd expect the IOPS to vary as we increase the size of the data block we're requesting from our Seagate Momentus drive.



So our 66 IOPS Seagate drive will in a SQL Server scenario (with 64KB block sizes) actually give us 64 IOPS when reading and 56 IOPS when writing.

The emphasis here is that when talking about IOPS (and of course comparing them), it is important to confirm the block sizes being tested and whether we are talking about reading or writing data. This is especially important for drives where the transfer times start playing a more significant role in the total time taken for the IO operation to be serviced.

As real-world IOPS values are detrimentally affected when I/O block sizes are considered (and also of course if we are writing instead of reading), manufacturers will generally quote a best case IOPS. This is taken from the time taken to *read* the minimum amount from a drive (512 bytes). This essentially yields an IOPS value derived from the drive's response time.

Cynicism aside, this simplified way of looking at IOPS is actually fine for ball-park values. Always worth bearing in mind that these quoted values are always going to be rather optimistic.

IOPS and Partial Stroking

If you recall, our 500GB Seagate Momentus has the following specs,

Spin Speed (RPM) .... 7200 RPM
Average latency 4.17ms
Seek time (Read) 11ms
Internal I/O data transfer rate 150MB/s
IOPS.......................................66

On the IOPS scale, we've already determined that this isn't exactly a performer. If we wanted to use this drive for a SQL database we'd likely be pretty dissapointed. Is there anything we can do once we've bought the drive to increase it's performance? Technically of course the answer is no, but strangely enough we can cheat the stats by being a little clever in our partioning.

To see how this works, let's partition the Momentus drive so that only the first 100GB is formatted. The rest of the drive, 400GB worth is now a dead-zone to the heads -they will never go there. This has a very interesting consequence to the drives seek time. The heads are now limited to a small portion of the drives surface, which means the time to traverse from one end of the formatted drive to the other is much smaller than the time taken it would have taken for the head to cross the entire disk. This reflects rather nicely on the drive's seek time over that 100GB surface, which has an interesting effect on the drive's IOPS.

To get some figures, let's assume that about 4ms of a drive's seek time is taken up with accelerating and decelerating the heads (2ms to accelerate, and 2ms to decelerate). The rest of the drive's seek time can then be said to be attributed to it's transit across the platter surface.

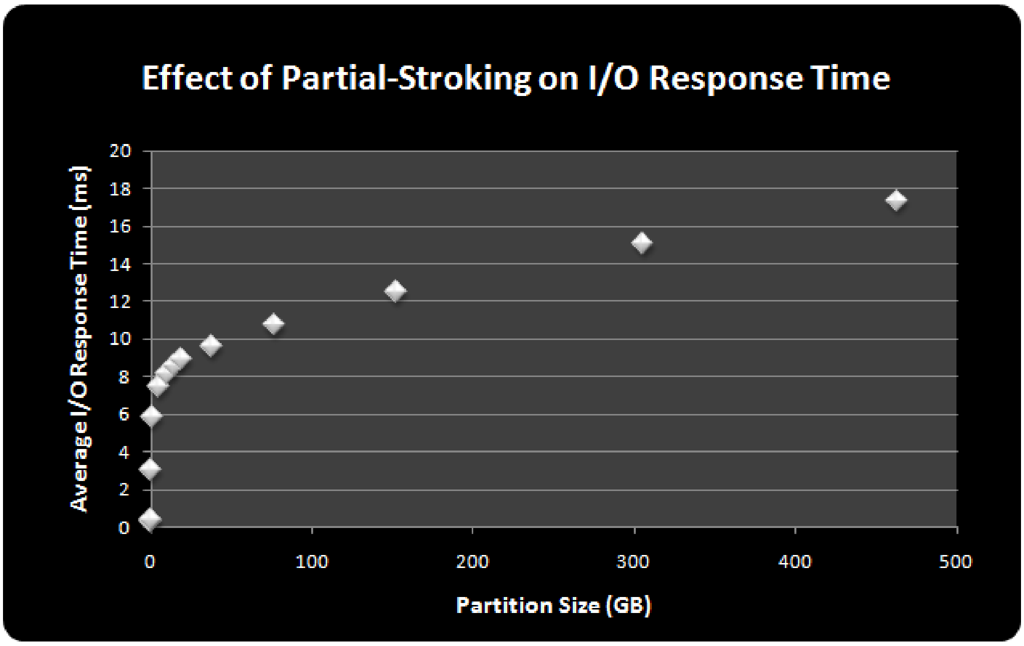
So, by reducing the physical distance the head has to travel now to a fifth of the drive's surface, we can estimate that the transit time is going to be reduced likewise. This results in a new seek time of (11-4)/5 + 4 = 6.4ms.

In fact, as more data is packed into the outside tracks due to ZBR this would be conservative estimate. If the latter four fifths of the drive were never going to be used, the drive stats would now look as follows,

Spin Speed (RPM) .... 7200 RPM
Average latency 4.17ms
Seek time (Read) ....6.4ms (for 0-100GB head movement restriction)
Internal I/O data transfer rate 150MB/s
IOPS.......................................94

The potential IOPS for this drive has increased by 50%. In fact, it's pretty much compariable now to a high-end 7200RPM drive! This trick is called *partial stroking,* and can be a quite effective way to ensure slower RPM drives perform like their big RPM brothers. Yes, you do lose capacity but in terms of cost you can save overall.

To see if this really works, I've used IOMETER to gather a number of response times for my Seagate Momentus using various partition sizes and a 512 byte data transfer.



Here we can see that the back of envelope calculation wasn't so bad -the average I/O response time here for a 100GB drive worked out to be 11ms and the quick calculation gave about 10.5ms. Not bad considering a lot of guess work was involved -my figures for head acceleration and deceleration were plucked out the air. Further I didn't add a settling time for the head before it started reading the data to allow the vibrations in the actuator arm to setting down. In truth, I likely over-estimated the arm accelleration and decelleration times which had the effect of absorbing the head settle time.

But, as a rough calculation I imagine this wouldn't be too far off for most drives.

Your milage will of course vary across drive models, but if for example you are looking at getting a lot of IOPS for a 100GB database, I'd expect that a 1TB 7200RPM Seagate Barracuda with 80 IOPS could be turned into a 120 IOPS drive by partitioning it for such a purpose. This would take the drive into the 10K RPM ballpark on the IOPS scale for *less than half the price* of a 100GB 10K RPM disk.

As you can see, this technique of ensuring most of the drives surface is a 'dead-zone' for the heads can turn a modest desktop harddisk into an IOPS king for its class. And the reason for doing this is not to be petty, or prove a point -it's *cost*. Drives with large RPMs and quoted IOPS tend to be rather expensive.

Having said that, I don't imagine though that many vendors would understand you wanting to effectively throw the bulk of your drives capacity out of the window. Your boss either...

How Many IOPS Do We Need?

Whilst enhancing our IOPS with drive stroking is interesting, what we're missing at the moment is where in the IOPS spectrum we should be aiming to target our disk subsystem infrastructure.
The ITSM 7.1 Planning and Implementation Guide has some interesting figures for a 20,000 node setup where SQL I/O was profiled for an hour at peak time,

SQL data file I/O per second

Metric	Value
Number of I/O per second.	238.7
Percent of write I/O per second.	98%
Percent of read I/O per second.	2%

TempDB database I/O per second

Metric	Value
Number of I/O per second.	1.3
Percent of write I/O per second.	49%
Percent of read I/O per second.	51%

Log files I/O per second

Metric	Value
Number of I/O per second.	593.8
Percent of write I/O per second.	100%
Percent of read I/O per second	0%

The conclusion was that the main SQL Server CMDB database required on average 240 write IOPS over this hour window. As we don't want to target our disk subsystem to be working at peak, we'd probably want to aim for a storage system capable of 500 write IOPS.

This IOPS target is simply not achievable through a single mechanical drive, so we must move our thinking to drive arrays in the hope that by *aggregating* disks we can start multiplying up our IOPS. As we'll see, it is at this point things get murky.....

IOPS, Disk Arrays & Write Penalties

A quick peak under the bonnet of most enterprise servers will reveal a multitude of disks connected to a special disk controller called a RAID controller. If you are not familiar with RAID, there is plenty of good online info available on this topic, and RAID's wikipedia entry isn't such a bad place to start.

To summarise, RAID stands for **R**edundant **A**rray of **I**ndependent **D**isks. This technology answers the need to maintain enterprise data integrity in a world where harddisks have a life expectancy and will someday *die*. The RAID controller abstracts the underlying physical drives into a number of *logical* drives. By building fault-tolerance into the way data is physically distributed, RAID arrays can be built to withstand a number of drive failures before data integrity is compromised.

Over the years many different RAID schemes have been developed to allow data to be written to a disk array in a fault tolerant fashion. Each scheme is classified and allocated a RAID *level*. To help in the arguments that follow concerning RAID performance, let's review now some of the more commonly used RAID levels,

- RAID 0
  This level carves up the data to be written into *blocks* (typically 64K) which are then distributed across all the drives in the array. So when writing a 640KB file through a RAID 0 controller with 5 disks it would first divide the file into 10 x 64KB blocks. It would then write the first 5 blocks to each of the 5 disks *simulateneously*, and then once that was successful proceed to write the remaining five blocks in the same way. As data is written in layers across the disk array this technique is called *striping*, and the block size above is referred to as the array's *stripe size*. Should a drive fail in RAID 0, the data is lost -*there is no redundancy*. As the striping concept used here is the basis of other RAID levels which do offer redundancy, it is hard to omit RAID 0 from the official RAID classification.

RAID 0's great benefit is that it offers a much improved I/O performance as all the disks are potentially utilised when reading and writing data.

- RAID 1
  This is the simplest to understand RAID configuration. When a block of data is written to a physical disk in this configuration, that write process is exactly duplicated on another disk. For that reason, these drives are often referred to as mirrored pairs. In the event of a drive failure, the array and can continue to operate with no data loss or performance degradation.
- RAID 5
  This is a fault tolerant version of RAID 0. In this configuration each stripe layer contains a parity block. The storing of a parity block provides the RAID redundancy as should a drive fail, the information the now defunct drive contained can be rebuilt on-the-fly using the rest of the blocks in the stripe layer. Once a drive fails, the array is said to operate in a *degraded* state. A single read can potentially require the whole stripe to be read so that the missing drive's information can be rebuilt. Should a further drive fail before the defunct drive is replaced (and rebuilt) the integrity of the array will be lost.
- RAID 6
  As RAID 5 above, but now *two* drives store parity information which means that two drives can be lost before array integrity is compromised. This extra redundancy comes at the cost of losing the equivlaent of two drives worth of capacity in the RAID 6 array (whereas in RAID 5 you lose the equivalent of one drive in capacity).
- RAID 10
  This is what we refer to as a nested RAID configuration -it is a stripe of mirrors and is as such called RAID 1 + 0 (or RAID 10 for short). In this configuration you have a stripe setup as in RAID 0 above, but now each disk has a mirrored partner to provide redundancy. Protection against drive failure is very good as the likelihood of both drives failing in any mirror simultenously is low.You can *potentially* lose up to half of the total drives in the array with this setup (assuming a one disk per mirror failure).

  With RAID 10 your array capacity is half the total capacity of your storage.

Below I show graphically examples of RAID 5 and RAID 10 disk configurations. Here each block is designated by a letter and a number. The letter designates the stripe layer, and the number designates the block index within that stripe layer. Blocks with the letter p index are parity blocks.

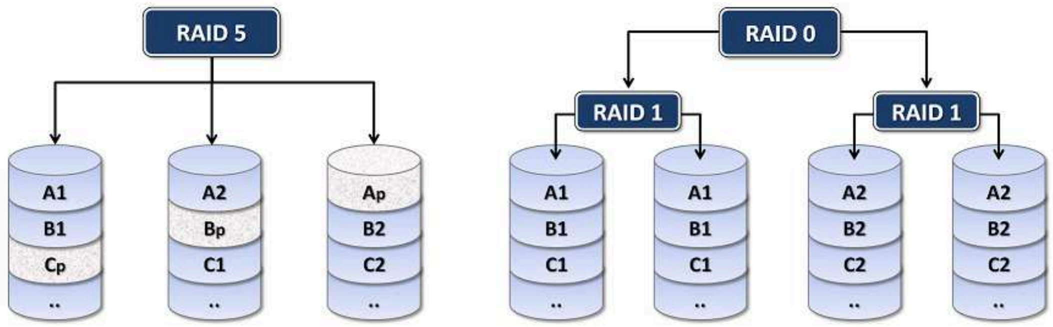


Illustration of two of the major RAID configurations in use today -RAID 5 (left) and RAID 10 (right)

As stated above, one of the great benefits that striping gives is performance.

Let's take again the example of a RAID 0 array consisting of 5 disks. When writing a file, all the data isn't simply written to the first disk. Instead, only the *first* block will be written to the *first* disk. The controller directs the *second* block to the second disk, and so on until all the disks have been written to. If there is still more of the file to write, the controller begins again from disk 1 on a new stripe layer. Using this strategy, you can simultaneously read and write data to a lot of disks, aggregating your read and write performance.

This can powerfully enhance our IOPS. In order to see how IOPS are affected by each RAID configuration, let's now discuss each of the RAID levels in turn and think through what happens for both incoming read and write requests.

- RAID 0
  For the cases of both read and write IOPS to the RAID controller, one IOPS will result on the physical disk where the data is located.
- RAID 1
  For the case of a read IOPS, the controller will execute one read IOPS on one of the disks in the mirror. For the case of a write IOPS to the controller, there will be two write IOPS executed -one to each disk in the mirror.
- RAID 5
  For the case of a read IOPS, the controller does not need to read the parity data -it just directs the read directly to the disk which holds the data in question resulting again in 1 IOPS at the backend. For the case of a disk write we have a problem - we also have to update the parity information in the target stripe layer. The RAID controller must therefore execute two read IOPS (one to read the block we are about to write to, and the other for obtain the parity information for the stripe). We must then calculate the new parity information, and then execute two write IOPS (one to update the parity block and the other to update the data block). One write IOPS therefore results in 4 IOPS at the backend!
- RAID 6
  As above, one read IOPS to the controller will result in one read IOPS at the backend. One write IOPS will now however result in 6 IOPS at the backend to maintain the two parity blocks in each stripe (3 read and 3 write).
- RAID 10
  One read IOPS sent to the controller will be directed to the correct stripe and one of the mirrored pair -so again only one write IOPS at the backend. One write IOPS to the controller however will result in two IOPS being executed in the backend to reflect that both drives in the mirrored pair require updating.

What we therefore see when utilising disk arrays is the following,

1. For disk reads, the IOPS capacity of the array is the number of disks in the array multiplied by a single drive IOPS. This is because one incoming read I/O results in a single I/O at the backend.
2. For disk writes with RAID, the number of IOPS executed at the backend is generally *not the same* as the number of write IOPS coming into the controller. This results the total number of *effective write IOPS* that an array is capable of being generally much less than what you might assume by naively aggregating disk performance.

The number of writes imposed on the backend by one incoming write request is often referred to as the RAID *write penalty.* Each RAID level suffer from a different write penalty as described above, though for easier reference the table below is useful,

RAID Level	Write Penalty
0	1
1	2
5	4
6	6
10	2

Knowing the write penalty each RAID level suffers from, we can calculate the effective IOPS of an array using the following equation,

$$IOPS_{eff} = \frac{n * IOPS}{R + (F * W)}$$

where *n* is the number of disks in the array, IOPS is the single drive IOPS, *R* is the fraction of reads taken from disk profiling, *W* is the fraction of writes taken from disk profiling, and *F* is the write penalty (or RAID Factor).

If we know the number of IOPS we need from our storage array, but don't know the number of drives we need to supply that figure, then we can rearrange the above equation as follows,

$$n = \frac{IOPS_{eff} * \left(R + (F * W)\right)}{IOPS}$$

So in our case of a SQL Server requiring 500 write IOPS (i.e. 0% READ pretty much) let's assume we are offered a storage solution of 10K SAS drives capable of 120 IOPS a piece. How many disks would we need to meet this write IOPS requirement? The table below summarises the results.

RAID Level	Write Penalty	n
0	1	4
1	2	8
5	4	16
6	6	25
10	2	8

What we see here is a HUGE variation in the number of drives required depending on the RAID level. So, your choice of RAID configuration is very, very important if storage IOPS is important to you.

I should say that most RAID 5 and RAID 6 controllers do understand this penalty, and will consequently cache as many write IOPS as possible, committing them during an idle window where possible. As a result, in real-world scenarios these controllers can perform slightly better than you'd anticipate from the table above. However once these arrays become highly utilised the idle moments become fewer which edges the performace back toward the limits defined above.

Summary

This finally then concludes today's article. I hope it's been useful and that you now have a better understand IOPS. The main points to take away from this article are,

1. Get involved with your server/storage guys when it comes to spec'ing your storage
2. The important measure for sequential I/O is disk throughput
3. The important measure for random I/O is IOPS
4. Database I/O is generally random in nature and in the case of the Altiris CMDB the SQL profile is also predominently write biased.
5. Choosing your storage RAID level is critical when considering your IOPS performance. By selecting RAID6 over RAID1 or 10 level you can potentially drop your total write IOPS by a factor of 3.

I should finish with an empahsis that this article is a starter on the disk performance journey. As such, this document should not be considered in isolation when benchmarking and spec'ing your systems. Note also that at the top of the reading list below is a *great* Altiris KB for SQL Server which will help you configure your SQL Server appropriately.

Next in the article pipeline (with luck) will be "Getting the Hang of Benchmarking" which will aim to cover more thoroughly what you can do to benchmark your systems once they are in place.

Good Luck!

Ian./

Further Reading

SQL Server 2005 and 2008 Implementation Best Practices and Optimization - A *great* symantec KB article on improving SQL Server performance

http://www.pcguide.com/ref/hdd - This is a great reference for how harddisks work. It includes everything you'd ever want to know about how harddisks work.

http://www.zdnet.com/blog/ou/how-higher-rpm-hard-drives-rip-you-off/322 - the blog entry which got me interested in drive stroking

http://www.seagate.com/staticfiles/support/disc/manuals/notebook/momentus/XT/100610268b.pdf -the Seagate Momentus specification sheet

http://vmtoday.com/2009/12/storage-basics-part-i-intro/ - A nice series of articles by Joshua Townsend on storage

http://www.seagate.com/docs/pdf/whitepaper/tp613_transition_to_4k_sectors.pdf -an interesting Seagate whitepaper on the transition from 512 Byte sectors to 4K sectors.

http://www.techrepublic.com/blog/datacenter/calculate-iops-in-a-storage-array/2182 -Scot Lowe's great TechRepublic article on IOPS and storage arrays

http://oss.oracle.com/~mkp/docs/ls-2009-petersen.pdf -Martin Peterson's paper on I/O topology