An Investigation in to iSCSI performance optimization for Virtualized Sequential SQL Database operations.

Abstract

Many protocols exist with the purpose of presenting a filesystem to compute nodes for use as storage within a data-center. FibreChannel is widely used in large businesses where performance is the biggest requirement; This has some disadvantages as it's expensive to implement, and requires its own cables to be run. Other solutions like FiberChannel over Ethernet (FCoE) and Advanced Technology Attachment over Ethernet (ATAoE) also exist. This paper will investigate the iSCSI protocol because it's often used in small businesses environments over the others, due to its relatively low cost to implement. However, iSCSI has the disadvantage of being slower for random accesses operations, relative to the other solutions. This paper will investigate different optimizations that can be made to improve web-server-responsiveness using an iSCSI file-system. Performance will be measured by querying a SQL-DB-server and measuring the average and max-time required to respond to a batch of requests. The optimizations that will be tested are physical connection distance between the iSCSI server and the client, network settings such as jumbo packets, CPU frequency, multipath IO and the use of a local memory-backed cache on the client. I propose a new solution for better random I/O performance by spreading the load between multiple iSCSI shares.

Introduction

In modern commuters, a complex cache hierarchy is used between it's storage and its processing, because modern CPUs will usually far outpace storage in terms of throughput. For example, even in simple consumer-grade systems, there can be 3 layers of cache between the CPU and the system main memory, and multiple layers between that and its storage; For example, HDDs (and even some newer SSD's https://www.anandtech.com/show/13078/the-intel-ssd-660p-ssd-review-qlc-nand-arrives) will have a read-write cache to mask the slow performance of the storage medium.

In virtualized infrastructure, this problem gets compounded, as CPU performance usually doesn't suffer, where as I/O latency is greatly increased. CenturyLink found that network latency between two servers in miroseconds for Kubernetes jumps from around 37-40microseconds on bare metal to around 120-135microseconds when virtualized. https://www.ctl.io/developers/blog/post/kubernetes-bare-metal-servers. For iSCSI performance, this would add additional latency to the storage, reducing its performance and the performance of applications that depend on it.

Despite all of the performance penalties that come with adding storage latency, iSCSI is still used within virtualized infrastructure greatly over other methods due to its flexibility with different solutions, maturity, vendor support, and performance. IBM created an article on "Storage to use with VMware Systems" where iSCSI was the most supported protocol in the storage solutions tested (4/6) and for the high performance "Tier 1 workloads", being supported in 2/3 of the solutions.

https://console.bluemix.net/docs/infrastructure/vmware/select-storage-option-use-vmware.html#storage-to-use-with-vmware-systems

Last year, Oracle released “Paravirtualized Block Volume Attachments" for VM’s In oracle cloud, which replaces iSCSI and so bypasses the need for network hardware virtualisation, as it passes disk-commands to the hypervisor through and API directly. However even with this, oracle say that iSCSI should still be used for larger block volumes to achieve maximum IOPS performance.

https://blogs.oracle.com/cloud-infrastructure/paravirtualized-block-volume-attachments-for-vms

For business and research applications such as machine learning, Big data, data archiving, or log-file processing where data-sets can be very large, access time increases even more, and drastically so with size, as indexing becomes a problem.

For certain applications such as SQL servers, file-system performance can greatly impact the completion time of a request when the database is unable to fit in to main memory. The article by Jonathan Foster for Microsoft demonstrates that fusionio drives (which were an PCI-Express-based ssd-like storage medium designed for high I/O workloads) shows that SQL server performance can be increased even over a SAS-HDD based RAID-0 setup by up to twice the speed at certain workloads. They also run the same set of tests on a newer and faster sever with many more processors and much more system memory, showing that in the majority of cases, the faster FusionIO storage had a larger benefit than the increased CPU an memory.

https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/hh240550(v=msdn.10)

Timothy A Wiseman found similar results in his testing by using a traditional SSD.

https://timothyawiseman.wordpress.com/2012/07/29/the-effects-of-an-ssd-on-sql-server-performance/

These articles show how by increasing storage performance can vastly improve SQL Database performance over increases to core count and the quantity of system memory.

Problem Statement

This paper will look at the different optimizations that can be made to iSCSI performance in a virtualized environment to benefit very large, sequential SQL Database operations. The parameters tested will include:

- iSCSI File-system block size
- The Effect of NIC’s capable of SR-IOV
- Using multiple iSCSI shares in a RAID-like configuration

Understanding of Known Solutions/Approaches

For virtualized SQL servers, having a dedicated NIC for a VM can greatly improve performance as there is no resource contention with other VMs. By assigning exclusive

access of a hardware device to a VM using PCI(-e) passthrough, the guest can take full control of the network card. This would result in near-bare metal performance, but does not lend to a virtual infrastructure, as you have now tied a physical piece of hardware to a virtual machine.

IBM Research-Haifa and Technion-Israel Institute of Technology looked in to how host the number of interrupts from I/O devices (such as network cards) affect performance of the devices themselves.

http://www.mulix.org/pubs/eli/eli.pdf

For example, if a hardware device needs to pass information to a guest OS, first, the guest must be paused and the host resume execution, handle the request from the hardware, and pass through a corresponding interrupt to the guest OS, and switch back to the guest OS. This process must be done in reverse once the guest has finished dealing with the hardware. The article suggests there are over 150,000 interrupts per second for high I/O workloads per core. One method they investigate is interrupt coalescing, where multiple interrupts are grouped up and dealt with at once, so there are less switches needed between guest and host. This comes with a latency penalty as requests will be added to a queue before being dealt with. The solution proposed by IBM and Technion is that all interrupts go to a second interrupt descriptor table inside the VM, that then routes the interrupts to the guest without the need for the CPU to change from guest to host. In their testing, this can achieve over 97% of bare metal sequential throughput of a Emulex OneConnect 10Gbnps NIC. This means for iSCSI traffic that this NIC could carry, there will be no guest-host switches at all for read-write operations.

Girish Motwani and K. Gopinath at the Indian Institute of Science, Bangalore, conduced some research in the mid 2000's for improving iSCSI throughput performance by using alternative TCP congestion control algorithms.

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.133.9623&rep=rep1&type=pdf

However, because of the age of the article, they were only using 100Mb/s network infrastructure, but this should still be applicable to today's high speed networks. Their results showed that using different TCP congestion algorithms can increase throughput drastically by properly utilizing the full capability of the link. One thing to note is that they forced a network latency of 40ms, which on a LAN is unusually large by today's standards.

A much more recent article by researchers at Ochanomizu University and Kogakuin University in Tokyo along-aside Softbank Telecom Corporation, also looked at sustained sequential isSCSI throughput with different different TCP congestion algorithms and on 1Gb/s networks and found that there is less difference between the algorithms on smaller latency networks.

https://pdfs.semanticscholar.org/2044/346a670601418ff214ecd557c9d05a95342e.pdf

This is because all the algorithms are able to evaluate the maximum throughput of the link quicker when the link has a lower latency. This time to evaluate is shown in Figure 6 of the paper where the difference in algorithms is much more pronounced when the test is shorter.

Girish Motwani and K. Gopinath also looked at how increasing the number of TCP connections can have on performance, and found that by using multiple, they were able to use more of the available bandwidth of the network. This is due to the congestion algorithm in each session not having to use the whole bandwidth of the network, and so it is able to increase the transfer speed to the maximum available bandwidth more quickly.

A paper by researchers at Colorado Center for Information Storage found the same results as the above two papers, where by increasing latency decreases sequential throughput.

https://pdfs.semanticscholar.org/b551/3c6ff13fb82bb816aee8afd3951c069c7f78.pdf

They also conducted some research that showed by increasing block size could improve sequential throughput, but only to 1024B block size, and only in certain scenarios. In this article they are also only able to achieve very low speeds because of poor performance of the disk and CPUs available at the time of the article. Because of the age of the article, its use today is questionable, however as the results do not contradict the newer articles, it seems safe to extrapolate results with caution, given what we currently know about the behavior of hardware today. They do include results for pure network transfer (without iSCSI) for 2Gb/s network cards, and show that increasing the block size even above 16KB can improve sequential performance. This should be less noticeable with modern CPU's as they are much faster at processing commands, but block performance could affect performance with 10Gb+ networks. The downside to increasing block size will be latency, as you begin grouping up more requests together, causing unnecessary queues. Research will have to be done in to the trade off between I/O latency and throughput for file-system performance for an SQL server.

The paper by researchers at Ochanomizu University and Kogakuin University in Tokyo along-aside Softbank Telecom Corporation also details basic optimization that can be made to iSCSI settings to help improve performance. Just as with different TCP settings, this can improve sequential throughput on higher latency networks, although the results are more drastic here, as after only about 5ms of round trip time, throughput is reduced to around 1/3 of the throughput available at <1ms. They continue their investigation and find that the iSCSI will send traffic in bursts, with gaps between the burst equal to the round trip time of the network. After modification of part of the linux kernel that handles socket connections, they are able to dramatically improve sequential performance on networks with a round trip time of >5-10ms. It should still be noted however that performance does not improve with networks of a RTT of <5ms, and at RTTs of over 20ms, sequential throughput is reduced to less than 2/3 of the throughput available at <1ms, although this is still a huge improvement.

Analysis and Support of your proposed solution

Some considerations can already be recommended, such as reducing the physical latency of the network between client and server. I will be testing the effect of changing the filesystem block size for the ISCSI volume, as the article by https://pdfs.semanticscholar.org/b551/3c6ff13fb82bb816aee8afd3951c069c7f78.pdf suggests that speed can be increased dramatically under certain loads. Along aside this, I will be testing the affect of SR-IOV as mentioned in the mulix ELI article (http://www.mulix.org/pubs/eli/eli.pdf) to reduce interrupts between the host and guest OS. Unfortunately, I will not be able to test the exact solution they suggested due to time constraints.

The new solution I propose is to have the load of a file-system spread across multiple iSCSI shares to lessen the load on a single share. This would be done in by striping multiple iSCSI shares in software to create one high performance volume.

Testing Methodology and Test Equipment:

For all results, the iSCSI server was comprised of the following equipment:

- Intel “core i5” 4590 CPU @ 3.5GHz
- 12GB DDR3 @ 1333Mhz
- 1+4 Gigabit Ethernet
- Windows Server 2016 Standard (version 1607, build 14393.2551)
- Sandisk 128GB SATA-6Gb/s SSD (SDSSDHP128G)
- Windows Server iSCSI target

Network Infrastructure:

- Network Switch was a 24-port Dell PowerConnect 2724
- 1m cat5e Cable between all links.

The iSCSI target was ran on an SSD as an SSD will be able to serve requests and will have a considerably shorter latency than a standard HDD, and will be able to saturate out GbE network more.

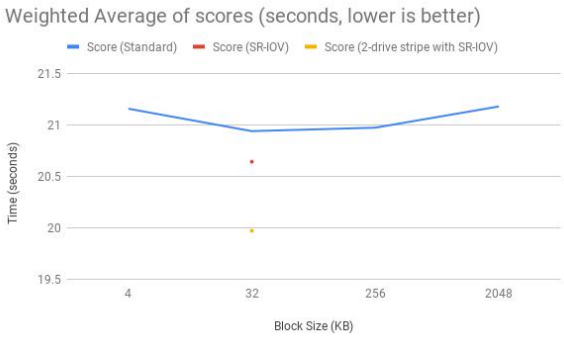
Test equipment software configuration and results:

Test Equipment:

- Windows 10 VM (Telemetry and monitoring disabled at install)
- 2x “Haswell” Cores @ 3.5GHz
- 4GB RAM (Static Allocation)
- 1GbE adapter (SR-IOV Compatible)

Results:

Standard Ethernet, 4KB Block							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	41.08	40.66	41.89	41.38	41.2525	0.25	10.313125
Select serial_number from drive_stats where failure = 1;	40.73	40.65	40.64	40.71	40.6825	0.25	10.170625
delete from drive_stats where date like "2017-01-%";	174.6				174.6	0.2	34.92
OPTIMITZE TABLE drive_stats;	779.4				779.4	0.05	38.97
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	27.7	28.88	28.88	28.8	28.565	0.4	11.426
						TOTAL	21.15995
Standard Ethernet, 32KB Block							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	41.83	42.03	41.8	42.41	42.0175	0.25	10.504375
Select serial_number from drive_stats where failure = 1;	41.52	41.53	40.97	41.05	41.2675	0.25	10.316875
delete from drive_stats where date like "2017-01-%";	165.72				165.72	0.2	33.144
OPTIMITZE TABLE drive_stats;	790.7				790.7	0.05	39.535
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	27.98	28.04	28.02	28.05	28.0225	0.4	11.209
						TOTAL	20.94185
Standard Ethernet, 256KB Block							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	41.05	41.12	41.11	41.11	41.0975	0.25	10.274375
Select serial_number from drive_stats where failure = 1;	41.05	40.97	41.59	42.47	41.52	0.25	10.38
delete from drive_stats where date like "2017-01-%";	159.7				159.7	0.2	31.94
OPTIMITZE TABLE drive_stats;	821.5				821.5	0.05	41.075
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	28.02	27.98	28.13	27.95	28.02	0.4	11.208
						TOTAL	20.975475
Standard Ethernet, 2048KB Block							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	41.7	41.66	43	42.86	42.305	0.25	10.57625
Select serial_number from drive_stats where failure = 1;	41.67	41.53	41.52	41.84	41.64	0.25	10.41
delete from drive_stats where date like "2017-01-%";	167.4				167.4	0.2	33.48
OPTIMITZE TABLE drive_stats;	802.3				802.3	0.05	40.115
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	28.23	28.28	28.35	28.42	28.32	0.4	11.328
						TOTAL	21.18185
Ethernet with SR-IOV, 32KB Block							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	41.52	41.98	41.94	41.44	41.72	0.25	10.43
Select serial_number from drive_stats where failure = 1;	41.34	41.44	41.5	43.42	41.925	0.25	10.48125
delete from drive_stats where date like "2017-01-%";	159.5				159.5	0.2	31.9
OPTIMITZE TABLE drive_stats;	784.4				784.4	0.05	39.22
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	27.84	28.14	27.96	27.96	27.975	0.4	11.19
						TOTAL	20.64425

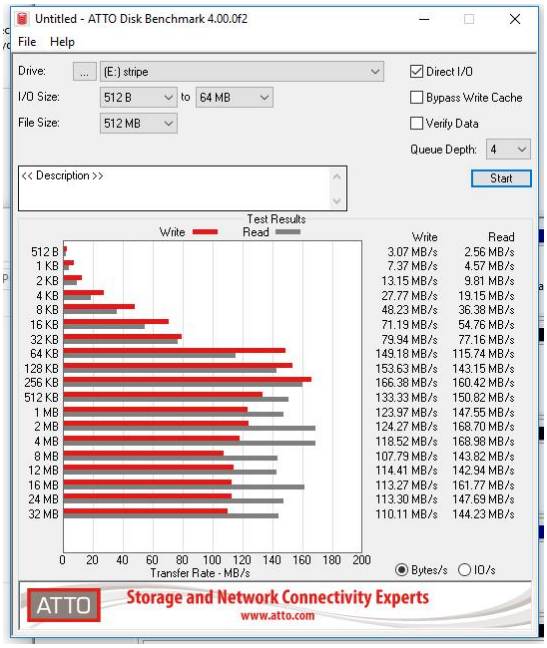


As the table shows, block size does have an effect on sequential performance, however not as much as predicted, as found in the https://pdfs.semanticscholar.org/b551/3c6ff13fb82bb816aee8afd3951c069c7f78.pdf article. This is likely due to the benefits of smaller blocks being negated (ie, faster fetch time) as all of the data of a particular section needs to be read in turn, rather than just parts of a section.

We can see that SR-IOV also helps to improve performance, however again not to the extent of as can be sen in the http://www.mulix.org/pubs/eli/eli.pdf paper. This is due to SR-IOV not being able to eliminate all of the switches between Guest OS and Host.

CPU usage was analyzed during the testing to see if that could be hindering the performance, but CPU usage was never observed to be above 70%, whereas disk usage was almost always at 100%.

The performance for the 2-drive stripe was much lower than expected, but still showed some improvements. Use of the network adapters for this test was around 40% at maximum, so I ran an ATTO disk benchmark to see if the iSCSI RAID was capable of the theoretical performance improvements.

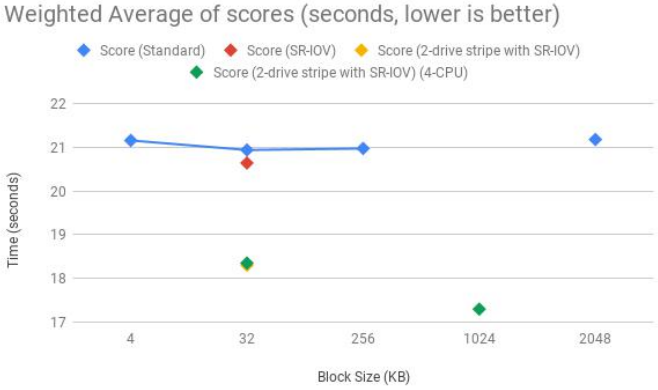


The results showed that greater than 70-80MB/s was possible on an NTFS file-system given the right block size.

It should be noted that CPU usage was much higher for the 2 iSCSI share SQL tests. I ran some further tests to verify if more CPU power could help improve performance, along with a more optimized block size.

2 iSCSI Shares, striped, 2CPU, 32KB							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	37.3	36.61	34.31	33.52	35.435	0.25	8.85875
Select serial_number from drive_stats where failure = 1;	33.69	33.54	33.63	33.27	33.5325	0.25	8.383125
delete from drive_stats where date like "2017-01-%";	161.2				161.2	0.2	32.24
OPTIMITZE TABLE drive_stats;	666.2				666.2	0.05	33.31
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	21.72	21.83	21.91	21.89	21.8375	0.4	8.735
						TOTAL	18.305375
2 iSCSI Shares, striped, 4CPU, 1024KB							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	31.54	30.49	31.02	30.95	31	0.25	7.75
Select serial_number from drive_stats where failure = 1;	30.67	30.86	30.56	30.6	30.6725	0.25	7.668125
delete from drive_stats where date like "2017-01-%";	158.8				158.8	0.2	31.76
OPTIMITZE TABLE drive_stats;	632.5				632.5	0.05	31.625
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	18.89	19.2	19.52	19.38	19.2475	0.4	7.699
						TOTAL	17.300425
2 iSCSI Shares, striped, 4CPU, 32KB							
Test	Run 1	Run 2	Run 3	Run 4	Average	Weights	Weighted Average
Select distinct count(serial_number) from drive_stats;	31.25	31	31.13	30.8	31.045	0.25	7.76125
Select serial_number from drive_stats where failure = 1;	30.86	30.89	31.14	31.33	31.055	0.25	7.76375
delete from drive_stats where date like "2017-01-%";	170.26				170.26	0.2	34.052
OPTIMITZE TABLE drive_stats;	647.9				647.9	0.05	32.395
select capacity_bytes from drive_stats order by capacity_bytes desc limit 3;	23.61	23.95	26.25	24.14	24.4875	0.4	9.795
						TOTAL	18.3534

Firstly, I doubled the core count of the SQL VM, and as you can see the 2 core (yellow) result is within margin of error of the 4 core result (green). Adding more cores for my tests does not increase performance, and is not the cause for a performance limit.



Increasing the block size to 1MB increased performance a noticeable amount. With a

software RAID of the iSCSI shares, windows interleaves the drives every 64KB. https://social.technet.microsoft.com/Forums/en-US/bca983c4-64d0-408a-943e-96345637e3b2/size-of-stripe-of-windows-server-software-raid?forum=winserver8gen

This means that at a block size of 32KB, only one drive is being accessed per request, removing the benifit of the interleave. With a large queue depth (multiple I/O queues), this becomes a non-issue as each drive will likely be getting blocks for different requests at the same time. When the block size is increased, both drives are required for every I/O transaction, and the full network path (both NICs in this experiment) are used.

Conclusions

In this paper, I looked at possible solutions to increase sequential SQL performance over an iSCSI share. Research showed that performance through tuning of the software through alterations such as below can lead to very dramatic performance increases.

- (In a virutalised environment)
  - Reducing host-guest context switches through grouping requests or modification of the hypervisor
- Using multiple TCP sessions per ethernet link
- Increasing of Block Size to reduce CPU load
- (In a high latency network (where RTT>5ms))
  - Altering or using different TCP Congestion control algorithms

My proposal was to use multiple Ethernet adapters each carrying a single iSCSI share that could be combined on the SQL server for increased performance. Results show that while performance can be increased, the increase is not linear as expected. For the full benefit of multiple links to be realized, it is likely that other modifications need to be made. Especially in relation to guest-host context switches, as the ELI research shows had the largest effect.

Further Research

Due to limited hardware, the SQL VM was on the same host as the iSCSI share. Separating these out to different machines should yield a performance increase, as the VM host has much less work to do, and even less context switches between the host and guest.

Testing on more modern and powerfull server-grade hardware should yield better results than the consumer equipment used in my testing.

My testing is limited as I only tested the hypothesis with one hyper-visor (Microsoft Hyper-V inside Windows Server 2016). With a hypervisor running on bare metal such as a discrete Hyper-V server, Xen, unRAID, or VMware ESXI, performance should be improved as the hypervisor has full access to the hardware, and is able to allocate it more efficiently than the Hyper-V-inside-windows-server-2016 approach I used.