The A to Z of SQL Server in the Cloud with Silk

Author: Kellyn Gorman, Director of Technical Advocacy, Silk

October 2023

Table of Contents

The Challenge with High IO in the Cloud

In today's rapidly evolving digital landscape, enterprises across the globe are seeking efficient and scalable solutions to manage and process their vast data reservoirs. SQL Server workloads, particularly those with high IO demands, stand out as prime candidates for migration to more adaptable and robust environments. Azure, with its versatile Infrastructure as a Service (IaaS) offering, emerges as an ideal platform for these migrations, providing both agility and reliability. However, a seamless transition and subsequent performance optimization require more than just a basic lift-and-shift strategy. It calls for an intricate integration of advanced storage solutions, like the network attached storage Silk.

This white paper delves deep into the intricacies of migrating high IO SQL Server workloads to Azure and Google Cloud (GCP) using IaaS, with a special emphasis on harnessing the power of network attached storage Silk. We aim to provide businesses with a comprehensive guide, highlighting best practices, potential pitfalls, and proven strategies to ensure a smooth transition and optimized post-migration performance. By understanding the synergy between Azure's IaaS and Silk's network storage capabilities, enterprises can truly unlock unprecedented scalability, efficiency, and speed for their SQL Server workloads.

Join us as we embark on this journey, providing a roadmap for businesses aiming to modernize their data infrastructure, ensuring not just a successful migration, but also a transformation that sets the foundation for future innovations and growth.

Metrics and Wait Events in SQL Server

Since its inception in the late 1980s as a joint project between Microsoft and Sybase, SQL Server has undergone profound transformations to become one of the premier relational database management systems (RDBMS) in the world. Over the decades, as organizations became increasingly data-centric, the demands on database systems surged. It was not just about storing and retrieving data anymore; performance, scalability, and efficiency became paramount. As SQL Server evolved to meet these challenges, there arose a need for more granular insights into its internal workings, leading to the development of advanced database metrics and workload information tools.

Wait events emerged as crucial indicators of performance bottlenecks within the SQL Server environment. They provided detailed insights into periods when threads in SQL Server processes were idle, typically due to resource contention or other blocking scenarios. By examining these wait events, database administrators could pinpoint the exact resources or operations causing delays, be it I/O operations, network latencies, or memory constraints. Over time, the analysis of wait events became a standard practice in performance tuning, enabling organizations to maximize the throughput of their SQL Server instances and ensuring smooth and responsive database operations.

The journey of SQL Server, from its early days to its current prominence, is not just a tale of technical advancements but also a reflection of the changing landscape of data management. The emphasis on database metrics and workload information, epitomized using wait events, underscores the industry's shift towards proactive performance management, ensuring that modern databases are not just repositories of data, but dynamic systems optimized for the demands of the contemporary digital age.

IO Wait Metrics in SQL Server

In the intricate world of SQL Server performance tuning, IO wait metrics stand out as pivotal indicators of I/O subsystem performance bottlenecks. Delving into these metrics, specific wait types such as PAGEIOLATCH, WRITELOG, and ASYNC waits emerge as particularly noteworthy. PAGEIOLATCH waits, for instance, signify delays experienced when reading data pages from disk into memory, often pointing to disk-related issues. WRITELOG waits, on the other hand, are associated with the time taken to write transaction logs to disk, a critical aspect of ensuring data durability. Meanwhile, ASYNC waits are linked to asynchronous operations, often related to backup, and restore processes. Together, these wait metrics provide database administrators with a comprehensive view into the I/O behavior of their SQL Server instances, enabling them to identify and address potential performance impediments effectively.

IO Specific Wait Events

- PAGEIOLATCH_EX/SH/UP
- WRITELOG
- ASYNC_DISKPOOL_LOCK
- ASYNC_NETWORK_IO
- ASYNC_IO_COMPLETION

Querying OS Wait Events Using DMVs

Since the introduction of SQL Server 2005, Dynamic Management Views (DMVs) have been instrumental in providing database administrators (DBAs) and developers with a window into the health, performance, and inner workings of SQL Server instances. These system views, coupled with Dynamic Management Functions (DMFs), allow users to obtain a wealth of information about server state, including query execution, index usage, wait statistics, and much more. Over the years, as SQL Server has evolved, so too have DMVs, with each release introducing new views and expanding the scope of available diagnostic information. Their consistent ability to offer real-time insights has solidified DMVs as an invaluable tool in the arsenal of SQL Server professionals around the world.

For our introduction to using DMVs with high IO SQL Server workloads, we'll begin with the sys.dm_os_wait_stats view:

```
SELECT *
FROM sys.dm_os_wait_stats dows
```

```
ORDER BY dows.wait_time_ms DESC;
```

```
| wait_type             | waiting_tasks_count | wait_time_ms | max_wait_time_ms | signal_wait_time_ms |
|-------------------------|---------------------|-------------|------------------|--------------------|
| LATCH_EX              | 12345               | 9876543     | 1234             | 567                 |
| CXPACKET              | 54321               | 8765432     | 4321             | 765                 |
| PAGEIOLATCH_SH        | 23456               | 7654321     | 2345             | 876                 |
| ...                   | ...                 | ...         | ...              | ...                 |
```

This query quickly offers us a peek into the most impactful wait events to the SQL Server. In our example output, note that PAGEIOLATCH_SH is the third in the list of top wait event consumers.

PAGEIOLATCH_*

In SQL Server, PAGEIOLATCH waits primarily indicate delays associated with reading data pages from disk into the buffer pool memory. Let's dive a little deeper into this:

What are these waits?

- PAGEIOLATCH waits occur when a query needs data that isn't already in the buffer pool (SQL Server's in-memory cache for data pages) and is forced to wait for the data page to be read from disk.

Types of PAGEIOLATCH waits:

- PAGEIOLATCH_SH: Wait to read a page into memory in a shared mode.
- PAGEIOLATCH_EX: Wait to read a page into memory in an exclusive mode.
- *UP/DT/KP/NL- Update/Destroy/Keep/Null Mode,* but the first two are the most common.

What do they indicate?

- High PAGEIOLATCH waits generally suggest that there might be an I/O subsystem bottleneck. It could mean that the storage system is not able to keep up with the read requests from SQL Server, leading to waits.
- It might also indicate that the working dataset of your queries is larger than what the buffer pool can hold, causing frequent reads from disk.

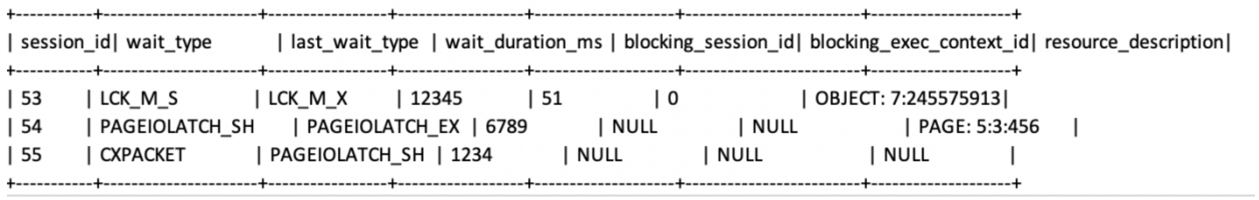
With PAGEIOLATCH waits in SQL Server, there is a clear indication of an IO problem and indicative of I/O-related waits when reading data pages from storage into memory. Addressing these waits often requires a combination of hardware upgrades and query/database optimization.

Checking for latches in Real-Time

Monitoring latches in real-time is crucial for maintaining the optimal performance and responsiveness of a SQL Server database system. Latches are lightweight synchronization primitives that protect in-memory structures from concurrent access. When excessive latch contention occurs, it can lead to queuing and delays, impacting the throughput and responsiveness of the system. By keeping an eye on latch behaviors in real-time, database administrators can swiftly detect potential bottlenecks and address them before they escalate into more severe performance degradation issues. Furthermore, real-time monitoring offers insights into the dynamic workloads and interactions happening within the server, allowing for proactive measures and immediate remediation, rather than post-event analysis. In essence, real-time latch monitoring serves as an early warning system, enabling timely interventions to ensure the database system's health and performance.

If a clear monitoring tool isn't at hand, having a query that can be run at the command line will suffice to inform the DBA of latching issues being experienced.

```sql
SELECT wt.session_id, wt.wait_type
, er.last_wait_type AS last_wait_type
, wt.wait_duration_ms
, wt.blocking_session_id, wt.blocking_exec_context_id,
resource_description
FROM sys.dm_os_waiting_tasks wt
JOIN sys.dm_exec_sessions es ON wt.session_id = es.session_id
JOIN sys.dm_exec_requests er ON wt.session_id = er.session_id
WHERE es.is_user_process = 1
AND wt.wait_type <> 'SLEEP_TASK'
ORDER BY wt.wait_duration_ms desc
```



```
+-----------+---------------------+----------------+-----------------+-------------------+------------------------+--------------------+
| session_id| wait_type           | last_wait_type | wait_duration_ms | blocking_session_id| blocking_exec_context_id| resource_description|
+-----------+---------------------+----------------+-----------------+-------------------+------------------------+--------------------+
| 53        | LCK_M_S             | LCK_M_X        | 12345           | 51                | 0                      | OBJECT: 7:245575913|
| 54        | PAGEIOLATCH_SH      | PAGEIOLATCH_EX | 6789            | NULL              | NULL                   | PAGE: 5:3:456      |
| 55        | CXPACKET            | PAGEIOLATCH_SH | 1234            | NULL              | NULL                   | NULL               |
+-----------+---------------------+----------------+-----------------+-------------------+------------------------+--------------------+
```

In this example:

- Session ID 53 is waiting for a shared lock (LCK_M_S) and its last wait type was for an exclusive lock (LCK_M_X). It has been waiting for 12,345 milliseconds and is being blocked by session ID 51.
- Session ID 54 is waiting on a shared page IO latch (PAGEIOLATCH_SH) with a previous wait type of exclusive page IO latch (PAGEIOLATCH_EX). It has been waiting for 6,789 milliseconds. There's no blocking session in this case.
- Session ID 55 is waiting due to parallelism (CXPACKET). Its last wait type was a shared page IO latch (PAGEIOLATCH_SH). It has been waiting for 1,234 milliseconds and is not being blocked by any session.

- Please note that the above is a fictitious example, and actual output will vary based on your SQL Server's activity and state.

WRITELOG

The SQL Server ecosystem revolves around ensuring that data remains consistent, durable, and readily available. One of the foundational elements of this ecosystem is the mechanism by which data changes are committed to disk. Here, the `WRITELOG` wait type plays a crucial role and, when not optimized, can have significant ramifications on the Input/Output (IO) performance of a system.

`WRITELOG` waits typically occur when the SQL Server is waiting for a log record to be flushed to the disk. Every transaction, when committed, is first written to the transaction log before it's hard-committed to the data file. This ensures data durability and is a core principle of the ACID (Atomicity, Consistency, Isolation, Durability) properties of transactions. However, when the system experiences elongated `WRITELOG` waits, it indicates that SQL Server is facing delays while attempting to write these transaction logs to the disk.

The direct implication of increased `WRITELOG` waits is degraded IO performance. If the SQL Server constantly queues up to write to the transaction log because of IO bottlenecks, it can cause a ripple effect, slowing down the processing of user queries and other essential operations. This is because the server must wait for the confirmation that a log record has been successfully written to disk before it can proceed with the next steps in the transaction process. Extended waits at this stage can compound and result in overall system sluggishness.

The root causes of prolonged `WRITELOG` waits can range from suboptimal disk subsystem performance, incorrect RAID configurations, or even network latency when dealing with network-attached storage. Addressing these waits requires a comprehensive approach, examining both the SQL Server configurations and the underlying hardware infrastructure. By doing so, one can ensure that the transaction log writes are swift and efficient, promoting optimal IO performance and guaranteeing the smooth operation of the SQL Server environment.

- As every change requires a change to the log buffer, followed by a write to the underlying disk, physical IO can be demanding.
- Caused by a transaction commit, unless set to be delayed durable for newer versions of SQL Server, (2014+)
- All writes and reads to log files, (LDF) are done sequentially and are written to disk. Write ahead logging forces the current log blog to the disk.
- Watch for waits on Always-on Availability Groups with remote log copy and HADR_SYNC_COMMIT waits
- Log block is also written when the sp_flush_log is executed in newer versions, (2014+)
- **Consider the location and storage type for LDF files carefully, ensuring it matches the demand.**

ASYNC Wait Events

SQL Server is a robust database management system with intricate mechanisms that ensure efficient data operations. Among these mechanisms, asynchronous (ASYNC) operations represent a category of processes that allow tasks to be executed without the need for immediate sequential processing. While ASYNC operations can lead to enhanced performance by not making the calling operation wait, they introduce their own set of wait types that, when prolonged, can have notable implications on the Input/Output (IO) performance of the system.

The primary ASYNC waits associated with IO include `ASYNC_IO_COMPLETION`, `ASYNC_NETWORK_IO`, and `ASYNC_DISKPOOL_WAIT`. These waits can provide valuable insights into potential IO bottlenecks within the SQL Server environment. For instance, `ASYNC_IO_COMPLETION` occurs when SQL Server is waiting for certain IO operations, like reading from data or backup files, to complete. Extended waits of this type could suggest inefficiencies in the disk subsystem or issues with the configuration of file systems. Similarly, `ASYNC_NETWORK_IO` waits point to delays in data transmission between SQL Server and the client application, indicating potential network bottlenecks.

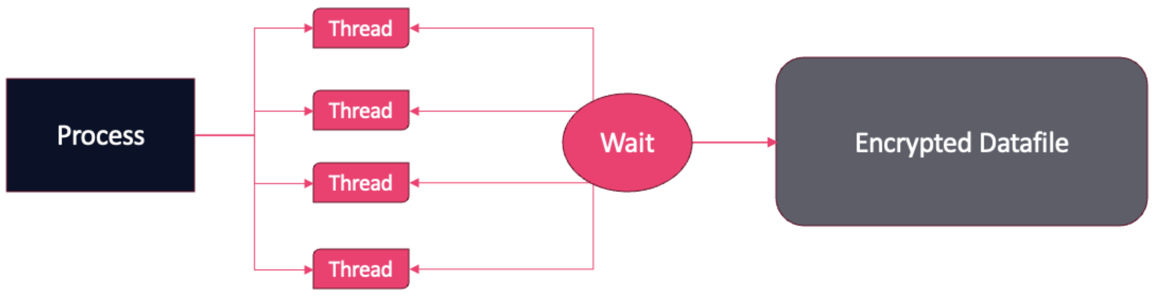
Prolonged ASYNC waits can significantly impact SQL Server's IO performance. For businesses relying heavily on real-time data access and processing, even minor IO inefficiencies can translate into substantial operational challenges. Identifying, understanding, and mitigating the causes of extended ASYNC waits is therefore paramount. These waits serve as a lens through which database administrators can diagnose underlying IO performance issues and make informed decisions about system optimizations.

In the realm of SQL Server optimization, recognizing the significance of ASYNC waits is the first step. The next, as detailed in this white paper, involves delving deeper into their root causes and strategizing solutions that ensure seamless and efficient IO operations.

ASYNC_DISKPOOL_LOCK

The first ASYNC wait event to be focused on will be ASYNC_DISKPOOL_LOCK, which occurs while SQL Server waits for threads to synchronize because of parallel disk operations, encrypting files with TDE and/or creating/deleting of files.

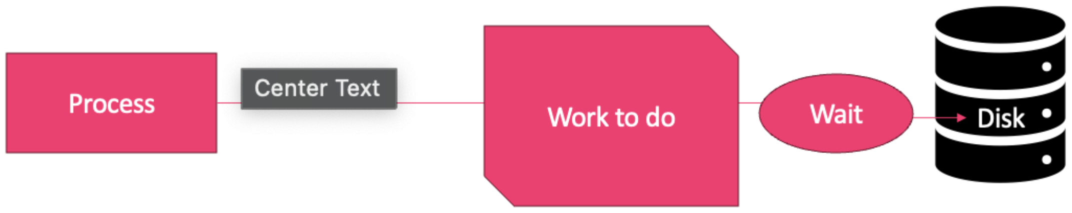
- Mostly maintenance operations- file create/grow, backup and restores.
- The larger the value, (over 50ms) the larger the concern



ASYNC_IO_COMPLETION

The ASYNC_IO_COMPLETION wait event is a non-datafile disk I/O completion. It occurs when growing a transaction log or IO that doesn't fit inside buffer pool.

- Can also be part of backup and restore.
- Slow storage can impact this wait event drastically.
- Measure the BACKUPIO timing to identify issues and consider the following:
  - Backup: faster backup storage.
  - Preallocate vs. smaller auto-growth for datafiles



ASYNC_NETWORK_IO

The network can be our last bottleneck and can also be impactful to relational workloads such as SQL Server.

- Data on application side can't process data being provided by SQL Server.
- Data inconsistencies in packet sizes between database(s) and applications.
- Consider reducing data sent over network by ensuring the right data is sent, (right vs. all)
- Verify Network Packet Size in connections strings is sized correctly.
- Consider faster network interface, (or larger pipe.)
- Break up applications using network interfaces, (use different Express route service, etc.)

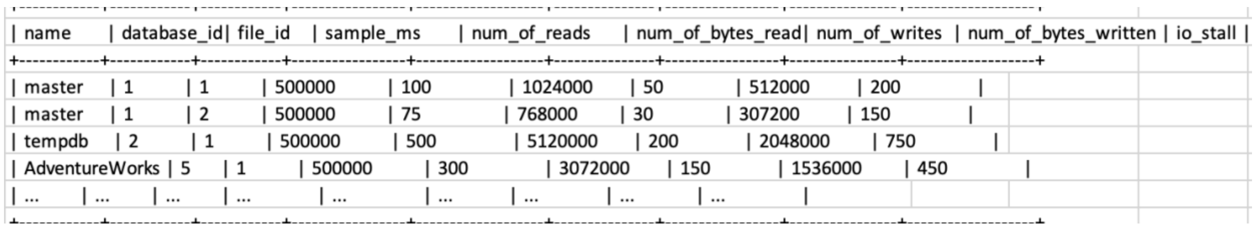
I/O Stalls

Although the I/O Stall wait event one might assume would be first, it will be covered last here in the list. This wait event directly refers and identifies I/O problems.

- Use the DMF dm_io_virtual_file_stats to view detailed info about stall times on data and/or log files.
- Requires the database ID and the database file ID to complete execution.

The following query is an example of how you might view information on I/O stalls in a SQL Server database:

```
select Db.name ,vfs.* from
  sys.dm_io_virtual_file_stats(NULL, NULL) AS VFS
  JOIN sys.databases AS Db
  ON vfs.database_id = Db.database_id;
```



```
| name        | database_id| file_id    | sample_ms      | num_of_reads      | num_of_bytes_read| num_of_writes  | num_of_bytes_written | io_stall |
+------------+------------+-----------+----------------+------------------+--------------+----------------+---------------+------------------+
| master     | 1          | 1          | 500000         | 100               | 1024000     | 50              | 512000        | 200          |
| master     | 1          | 2          | 500000         | 75                | 768000      | 30              | 307200        | 150          |
| tempdb     | 2          | 1          | 500000         | 500               | 5120000     | 200             | 2048000       | 750          |
| AdventureWorks | 5      | 1          | 500000         | 300               | 3072000     | 150             | 1536000       | 450          |
| ...        | ...        | ...        | ...            | ...               | ...         | ...             | ...           | ...          |
+------------+------------+-----------+----------------+------------------+--------------+----------------+---------------+------------------+
```

Tools and the DBA

All the data collected in the previous sections were done via Dynamic Management Views (DMVs) in SQL Server. This data is cached and cleared upon a cycle of the database or host and requires a workload worthy of measuring. This limitation must be acknowledged and the workaround for it is to use long term data collection via a secondary monitoring tool that tracks workload usage over time.

For SQL Server professionals, there are several common third-party tools used and simply may not have been configured or viewed for the same data we queried via the DMVs.

Some of the most popular tools are:

- Redgate SQL Monitor
- SolarWinds SQL Sentry
- Azure Monitor
- New Relic
- Quest Foglight

Furthermore, there are some internal tools that may provide similar value if used correctly:

- Extended Events
- Data Collector
- Performance Monitor

Dynamic Management View Data for Workload Information

To collect basic workload information about a SQL Server database via the Dynamic Management Views (DMVs) and functions, we'll begin with the following:

- **sys.dm_io_virtual_file_stats**
- **sys.fn_virtualfilestats**

To fulfill the arguments for the calls, you will need to identify the DB_ID and the File_ID you're interested in collecting the information on:

Locate the Database Info:

```
SELECT DB_ID('proddb') AS [DB ID]
GO
Sp_helpdb 'proddb'
```

```
GO
```

Once you have this information, you can then execute the calls on the function and the view to view the stats on the database files:

Usage:

```
sys.fn_virtualfilestats ( { db_id | NULL } , { file_id | NULL } )
```

Example:

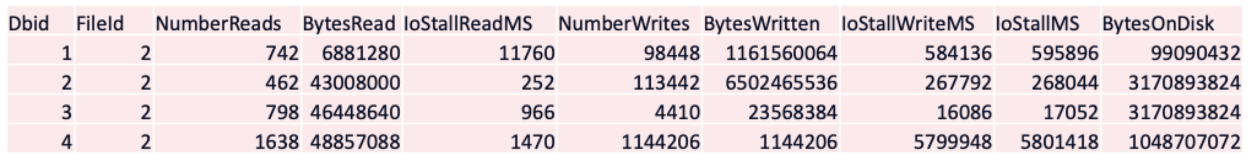
```
SELECT * FROM fn_virtualfilestats(2, NULL)
GO
```

Usage:

```
sys.dm_io_virtual_file_stats ( { db_id | NULL }, { file_id | NULL })
```

Example:

```
SELECT * FROM sys.dm_io_virtual_file_stats(2, NULL)
GO
```



Dbid	FileId	NumberReads	BytesRead	IoStallReadMS	NumberWrites	BytesWritten	IoStallWriteMS	IoStallMS	BytesOnDisk
1	2	742	6881280	11760	98448	1161560064	584136	595896	99090432
2	2	462	43008000	252	113442	6502465536	267792	268044	3170893824
3	2	798	46448640	966	4410	23568384	16086	17052	3170893824
4	2	1638	48857088	1470	1144206	1144206	5799948	5801418	1048707072

IO Usage at the Database Level

We can build out further on this information and present the data in a more usable format and provide exactly what we need vs. what the developer of the DMV/DMF may have foreseen it used for by writing our own query with it incorporated into it.

```
WITH IO_Per_DB
AS
(SELECT
 DB_NAME(database_id) AS Db
, CONVERT(DECIMAL(12,2), SUM(num_of_bytes_read + num_of_bytes_written)
/ 1024 / 1024) AS TotalMb
FROM sys.dm_io_virtual_file_stats(NULL, NULL) dmivfs
GROUP BY database_id)
 SELECT
Db ,TotalMb ,CAST(TotalMb / SUM(TotalMb) OVER() * 100 AS DECIMAL(5,2))
AS [I/O]
FROM IO_Per_DB
ORDER BY [I/O] DESC;
```

The above query now displays total MBPs and percentage of IO across entire SQL Server per database.

	Db	TotalMB	I/O%
1	ProdDB	431.00	5.42%
2	ProdDW	7248.00	91.19%
3	tempdb	269.00	3.38%

If we need to few this in a more detailed perspective, i.e. from the datafile view vs. the database, ensuring we identify prospective high IO areas that could become a bottleneck, we could use the following query:

```
WITH IO_Per_DB_Per_File
AS
(SELECT DB_NAME(dmivfs.database_id) AS Db
, CONVERT(DECIMAL(12,2), SUM(num_of_bytes_read + num_of_bytes_written)
/ 1024 / 1024) AS TotalMb
, CONVERT(DECIMAL(12,2), SUM(num_of_bytes_read) / 1024 / 1024) AS
TotalMbRead
, CONVERT(DECIMAL(12,2), SUM(num_of_bytes_written) / 1024 / 1024) AS
TotalMbWritten
, CASE WHEN dmmf.type_desc = 'ROWS' THEN 'Data File' WHEN
dmmf.type_desc = 'LOG' THEN 'Log File' END AS DataFileOrLogFile
FROM sys.dm_io_virtual_file_stats(NULL, NULL) dmivfs
JOIN sys.master_files dmmf ON dmivfs.file_id = dmmf.file_id AND
dmivfs.database_id = dmmf.database_id
GROUP BY dmivfs.database_id, dmmf.type_desc)
SELECT Db , TotalMb , TotalMbRead , TotalMbWritten , DataFileOrLogFile
, CAST(TotalMb / SUM(TotalMb) OVER() * 100 AS DECIMAL(5,2)) AS [I/O]
FROM IO_Per_DB_Per_File
ORDER BY [I/O] DESC;
```

	Db	TotalMb	TotalMbRead	TotalMbWritten	DataFileOrLogFile	I/O%
1	ProdDB	431	398	33	Data File	3.41%
2	ProdDB	42	6	36	Log File	1.12%
3	ProdDW	7248	4902	2346	Data File	79.84%
4	ProdDW	231	55	176	Log File	3.33%
5	tempdb	269	147	123	Data File	3.38%

With this change to the datafile level from viewing the IO at the database level, the differences in how much IO is created at the datafile level for the ProdDW (Data Warehouse) the distribution of reads vs. writes, knowing that writes are more challenging IO and it’s a third of the total IO on the datafiles. We can also see that the pressure on the transaction log for the data warehouse is surprisingly small, meaning that the batch loads have been written efficiently for such a high IO workload.

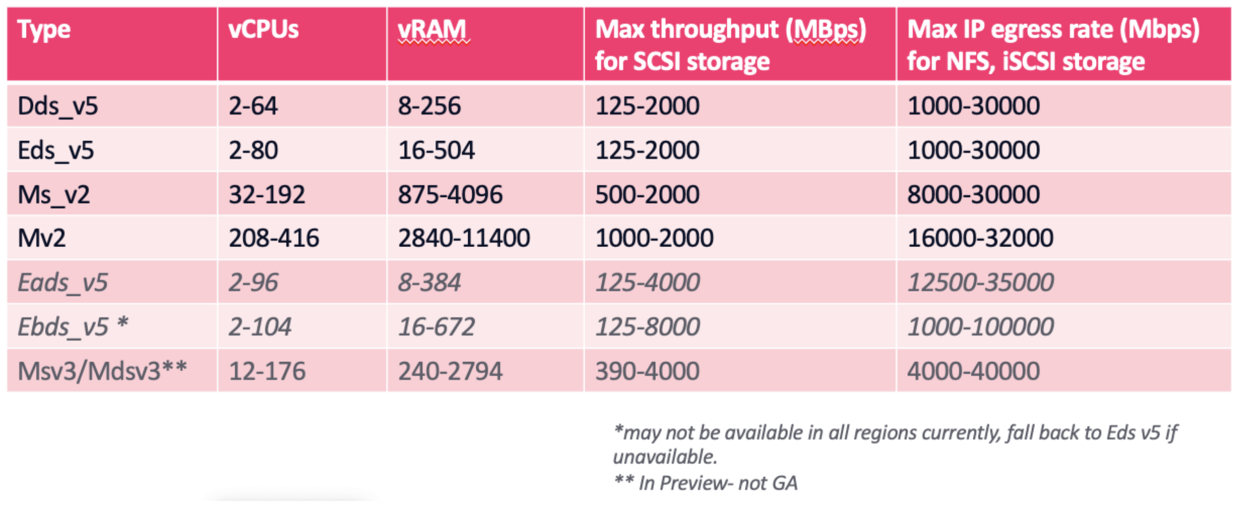
Armed with this information, we now can learn what features provided by major cloud vendors will support standard vs. high IO workloads.

How to High IO in the Cloud

Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) are both cloud computing services that come with certain restrictions to ensure resources are not over-allocated. These services are generally suitable for handling smaller workloads effectively. However, when it comes to managing larger, more demanding workloads, they often fall short. On the other hand, IaaS has the potential to handle the most extensive workloads, those that don't align with PaaS and Software as a Service (SaaS) solutions, but this is only achievable with the right knowledge and approach.

A and B-series are not suitable for high IO workloads and should be avoided. While D-series can be a viable option for certain tasks, it's advisable to match the SKU series to the production environment, albeit with reduced resources. On the other hand, L and H-series stand as anomalies when it comes to database workloads. However, they can be efficient if redundancy is integrated into the solution, but caution is paramount.

When architecting SQL Server into the cloud, it's crucial to first identify the specific workload needs. The D-series is typically utilized for general purposes. In the database industry, E-series, and M-series VMs are the most prevalent. Notably, E-series boasts the current highest IO limits in Azure. Meanwhile, the M-series is especially tailored for VLDBs, which refers to very large databases, or scenarios with intensive memory usage. However, it's essential to note that both E-series and M-series, despite their capabilities, will still necessitate adherence to recommended practices to effectively handle HIGH IO workloads.



Type	vCPUs	vRAM	Max throughput (MBps) for SCSI storage	Max IP egress rate (Mbps) for NFS, iSCSI storage
Dds_v5	2-64	8-256	125-2000	1000-30000
Eds_v5	2-80	16-504	125-2000	1000-30000
Ms_v2	32-192	875-4096	500-2000	8000-30000
Mv2	208-416	2840-11400	1000-2000	16000-32000
Eads_v5	*2-96*	*8-384*	*125-4000*	*12500-35000*
*Ebds_v5 **	*2-104*	*16-672*	*125-8000*	*1000-100000*
Msv3/Mdsv3**	12-176	240-2794	390-4000	4000-40000

**may not be available in all regions currently, fall back to Eds v5 if unavailable.*
*** In Preview- not GA*

Reducing Licensing Through Constrained vCPU VMs

Utilizing constrained vCPU VMs in Azure offers several advantages, particularly for database and application workloads. One of the primary benefits is the capability to isolate vCPU specifically for application licensing. This tailored allocation can be effectively matched with existing series VMs as outlined in the Azure Pricing Calculator. Furthermore, these VMs provide flexibility in storage management. Users can efficiently share storage between various databases or applications by leveraging network storage options, such as Silk.

These specialized VM are available in the D, E and M-series skus and are easily recognizable by the dash in the SKU name. The naming convention identifies the larger chassis memory and IO configuration reference to the left of the dash and the available vCPU to the right of the dash in the SKU:

Name	**vCPU**	**Specs**
Standard_E4-2ds_v5	2	Same as E4ds_v5
Standard_E8-4ds_v5	4	Same as E8ds_v5
Standard_E8-2ds_v5	2	Same as E8ds_v5
Standard_E16-8ds_v5	8	Same as E16ds_v5
Standard_E16-4ds_v5	4	Same as E16ds_v5
Standard_E32-16ds_v5	16	Same as E32ds_v5
Standard_E32-8ds_v5	8	Same as E32ds_v5
Standard_E64-32ds_v5	32	Same as E64ds_v5

The naming convention for constrained VMs in Azure can be misleading. While they are termed "vCPU constrained", the counts represent the only available vCPUs on the VM and in no way do customers have access to more vCPUs. Although there is a focus on the limited vCPU count on these machines, it's crucial to carefully match workloads based on IO and memory to ensure optimal performance and resource allocation, too.

How Storage Is Decided by the VM

For the cloud to infinitely scale for as many customer workloads as possible, limits need to be posed at some level. For all major cloud vendors, this is done at the compute or service level tier. As this white paper is focused on IaaS, we will focus on compute, but also mention that IaaS compute limits are higher on average than service level limits in almost every category. Even big data and noSQL services have lower IO limits they're able to achieve for every 1TB allocated than every corresponding infrastructure option.

With the introduction of partner solutions, often created to solve challenges such as high IO, the difference between what PaaS or SaaS can achieve vs. IaaS becomes greater.

diagram to help demonstrate.

table to show limits for Azure SQL for 1TB, CosmosDB per every 1TB, etc.? *

IOPS Can Mask Performance Issues
There are three main measurements that are used to judge performance for IO:

- IO Requests, (IOPS)
- Throughput, (MBPS)
- Latency

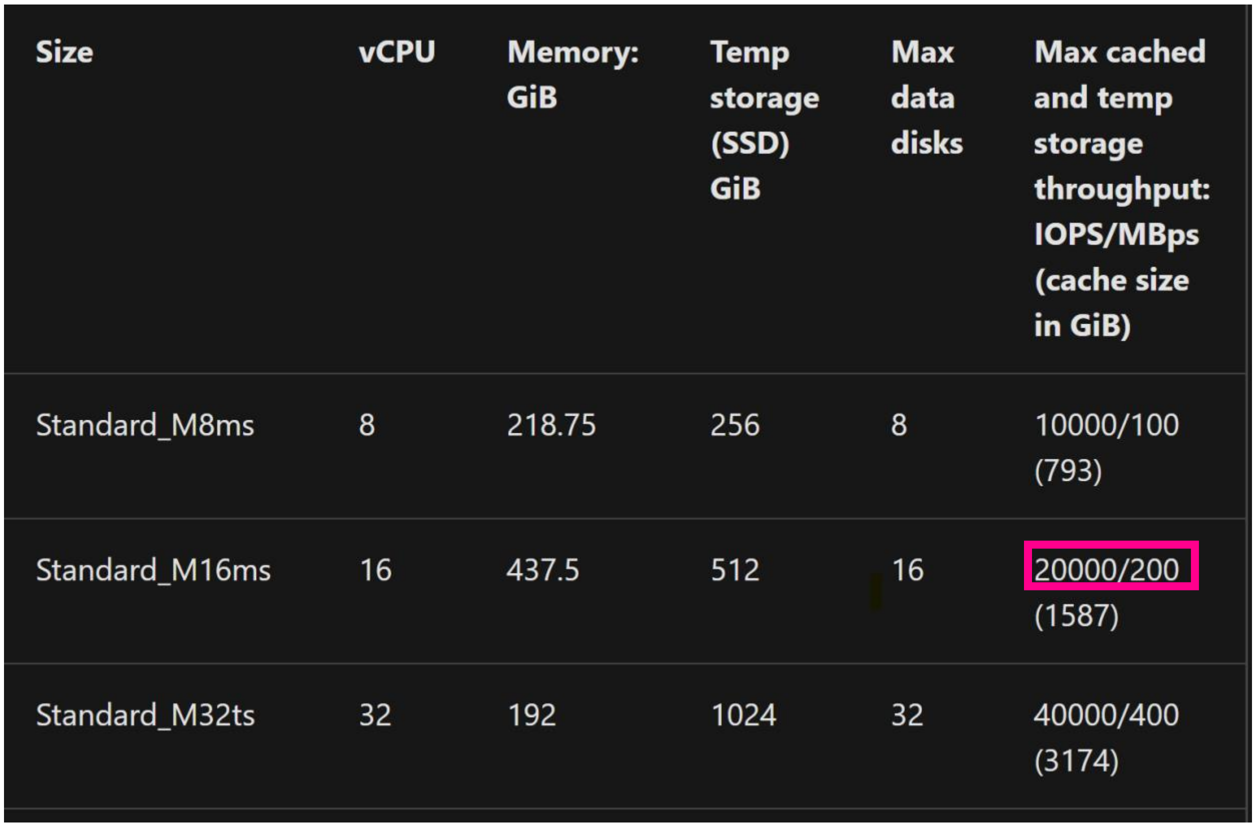
Once of the biggest challenges for relational workloads such as SQL Server is that request sizes can be in various sizes, a skewing performance challenges if IOPS is the only measurement used for identifying IO. IOPS is an attractive measurement for storage company marketing due to the higher overall numbers in demonstrations.   The challenge comes when a high IO situation such as nighttime batch loads and backups with low requests, but high throughput is the challenge that often must be overcome, and it can't be met because the storage provider didn't understand how important throughput and latency is to a relational system.

Type	**High**	**Average**	**Low**
IOPS	1 million/sec	300 K/sec	60 K/sec
MBPs	10GBPs	500MBps	25MBps
Latency	<7ms	<4ms	<1.5ms

Recognize that unlike the other two measurements, latency is better with a lower value.

So how does the VM impact the storage performance? On-premises, storage is striped, or storage arrays are connected, and the IO performance is separate, but in the cloud, to ensure that no one workload can overwhelm the available resources, a limit is set at the compute VM level. This can be viewed for each cloud vendor in their documentation and should always be considered when choosing a virtual machine.

A customer can allocate and stripe 10000MBPS throughput of ultra disk, but the VM will throttle the performance back to the upper limit allowed for that VM SKU.



Size	vCPU	Memory: GiB	Temp storage (SSD) GiB	Max data disks	Max cached and temp storage throughput: IOPS/MBps (cache size in GiB)
Standard_M8ms	8	218.75	256	8	10000/100 (793)
Standard_M16ms	16	437.5	512	16	20000/200 (1587)
Standard_M32ts	32	192	1024	32	40000/400 (3174)

As shown in the above information from Azure documentation, the M16ms has 16 vCPU, over 437 GiB of memory, but can only achieve 20000 IOPs and 200 MBPS. While this VM host would be a great option for a customer who needed high vCPU with high memory, a high IO workload using attached storage, such as premium SSD or ultra disk would throttle at only 200MBPS.

To understand how we would solve an example challenge of an imaginary customer facing this scenario- Using 16 vCPU and 437GiB of memory, but in need of much higher IO limits, how would we solve this?

Although the new M-series V3 are in private preview, the customer needs an answer now and this is where Silk shines.

In a previous section we discussed the importance of constrained vCPU VMs. E-series VMs have less memory with high IO, but we need the right combination. One that comes close to the following:

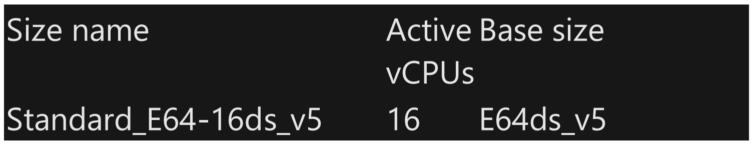
1. 16 vCPU to keep the CPU resource needs addressed, but not more that will drive the cost of the VM and the SQL Server licensing to increase.
2. Need at least 437 GiB of memory.
3. In our example, the customer thought they were going to get 10000MBPs of performance from Ultra Disk they striped.

High level, this solution, deployed to EastUS for just the VM and UltraDisk would be:

Resource	Cost
M16ms (Max IO is 200MBPs)	2243.29
Ultra Disk (says maxed at 4000MBPs)	30064.69
Total	**32307.98**

This was the high-level cost per month and would throttle at 200MBPs. No amount of scaling with the current VM series or storage could solve the problem, as both the VM series and the storage type is wrong for the workload.

I focused on the following constrained vCPU VM:



Size name	Active vCPUs	Base size
Standard_E64-16ds_v5	16	E64ds_v5

To translate the SKU name, (Size Name) we can demonstrate the following:

Type	Series	Chassis	vCPU Avail	Sub Letter	Version
E64-16ds_v5	E	64	16	S	V5
Enterprise class relational VM	E-series memory optimized	E64ds v5 chassis	16 vCPU	Premium SSD OS avail	Version 5, Ice Lake Chip

Why did I choose this one?

1. It will meet the customers need for 16 vCPU and SQL Server licensing.
2. Using the E-series detailed documentation for the Eds v5 series, we can see the following:

The chassis has:

- 512 GiB of memory
- Availability for a Premium SSD OS Disk
- Can have upwards of 32 attached disks, but will most likely not use them- why?
- Although the attached disk IO will top out at 1735MBPS without bursting and 3000MBPS with bursting, the network limits are more important.
- Only Egress on writes is limited to 30000Mbps, (notice that little ‘b’ there) with reads unlimited.
- For this example, we’ll use Silk Express

Resource	Cost
E64-16sv5	$2943.36

With this VM, we can deploy the database onto a Silk DataPod using Silk’s with PV2, which can be done for < $30K and we end up with the same price, but able to meet the actual workload and get extra resources for growth!

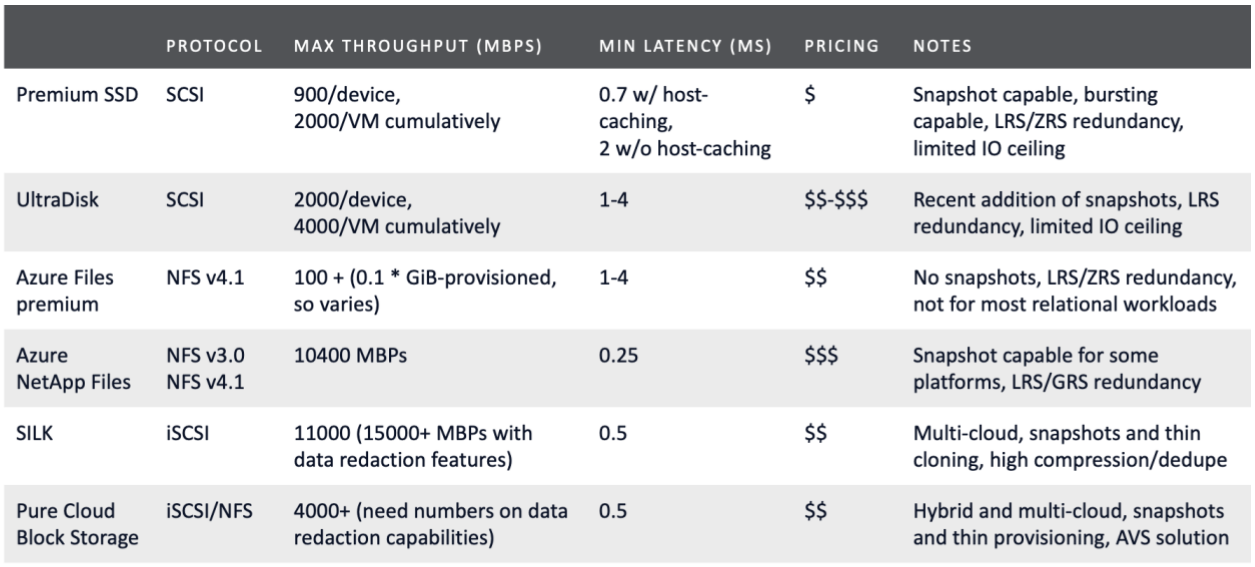
Storage Limits Create Bottlenecks

Attached storage, such as premium SSD, Premium SSD V2 and Ultra Disk all have limits set at the VM level that should be reviewed carefully before chosen to ensure they meet the demands of the workload they will support. These limits are outside the overall limits stated by storage documentation.

Cloud	Storage Type	Storage Documentation Limit	VM Hard Upper Limit	Difference
Azure	Premium SSD	9000MBPs	750MBPs	
Azure	PV2	1200MBPs	1000MBPs	
Azure	Ultra Disk	4000MBPs	2000MBPs	
GCP	HyperDisk	5000MBPs	5000MBPs	
Azure	Pure CBS	4000MBPs	4000MBPs	
Azure	ANF	10400MBPs	10400MBPs	
Azure GCP	CloudTapNetApp Files	15000MBPs	12500MBPs	
AzureAzure	Silk EnterpriseCloudTap	12500MBPs15000MBPs	15000MBPs12500MBPs	
AzureAzure	Silk PV2Silk Enterprise	7000MBPs12500MBPs	7000MBPs15000MBPs	
GCPAzure	Silk EnterpriseSilk PV2	12000MBPs7000MBPs	12000MBPs7000MBPs	

*Limits experienced in testing and running workloads/POC with customers.

Realistic Per VM Limits for Storage by Type



	PROTOCOL	MAX THROUGHPUT (MBPS)	MIN LATENCY (MS)	PRICING	NOTES
Premium SSD	SCSI	900/device, 2000/VM cumulatively	0.7 w/ host-caching, 2 w/o host-caching	$	Snapshot capable, bursting capable, LRS/ZRS redundancy, limited IO ceiling
UltraDisk	SCSI	2000/device, 4000/VM cumulatively	1-4	$$-$$$	Recent addition of snapshots, LRS redundancy, limited IO ceiling
Azure Files premium	NFS v4.1	100 + (0.1 * GiB-provisioned, so varies)	1-4	$$	No snapshots, LRS/ZRS redundancy, not for most relational workloads
Azure NetApp Files	NFS v3.0 NFS v4.1	10400 MBPs	0.25	$$$	Snapshot capable for some platforms, LRS/GRS redundancy
SILK	iSCSI	11000 (15000+ MBPs with data redaction features)	0.5	$$	Multi-cloud, snapshots and thin cloning, high compression/dedupe
Pure Cloud Block Storage	iSCSI/NFS	4000+ (need numbers on data redaction capabilities)	0.5	$$	Hybrid and multi-cloud, snapshots and thin provisioning, AVS solution

Understanding the drastic difference in the sheer throughput differences for some storage options is due to the storage connection to the VM. Storage solutions able to connect via the NIC, (network) as demonstrated in the VM section of the white paper, demonstrates how valuable network attached storage is.

Many of the solutions reviewed as part of this Silk white paper also have additional features as part of the solution to provide more value to the total cost of ownership (TCO) such as data redaction, data protection and snapshot capabilities.

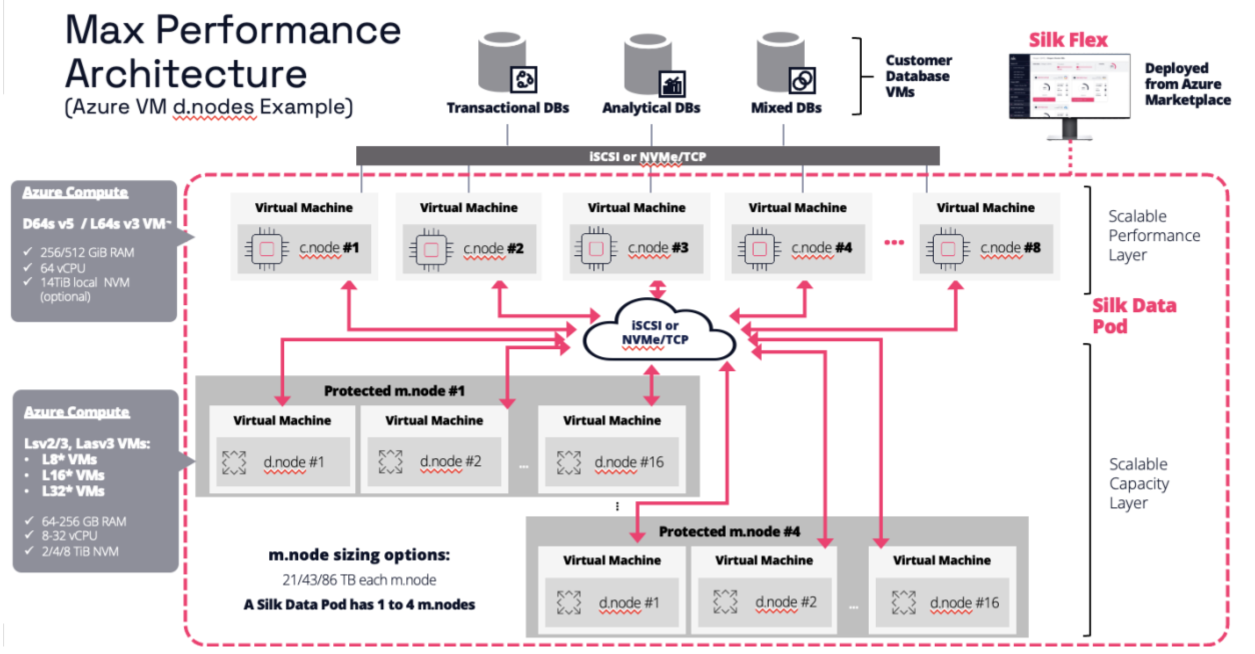
How Silk Wins with SQL Server

As the premier solution to provide the highest IO in Azure of all, it's essential to view the holistic platform that Silk provides for relational workloads, like SQL Server.

Architecture

Silk has two architecture deployments- Both deploy using Azure infrastructure under the covers and use Silk patented solutions and software. The first, architected for max performance, uses D and L series VMs with Kubernetes to create a resilient, self-healing Silk data pod, presented as a simple storage layer to the one or more VMs. This is an ingenious architecture design that creates an easily managed/presented, high IO opportunity which satisfies both scalable performance and capacity in cleanly separated layers called C nodes (performance) and m/d nodes (capacity).

Max Performance Architecture



The above architecture can be attached to multiple VMs and achieve over 25GBPS. Resiliency and reliability to build into the platform which wouldn't be available if built on the native VMs, nor would the performance.

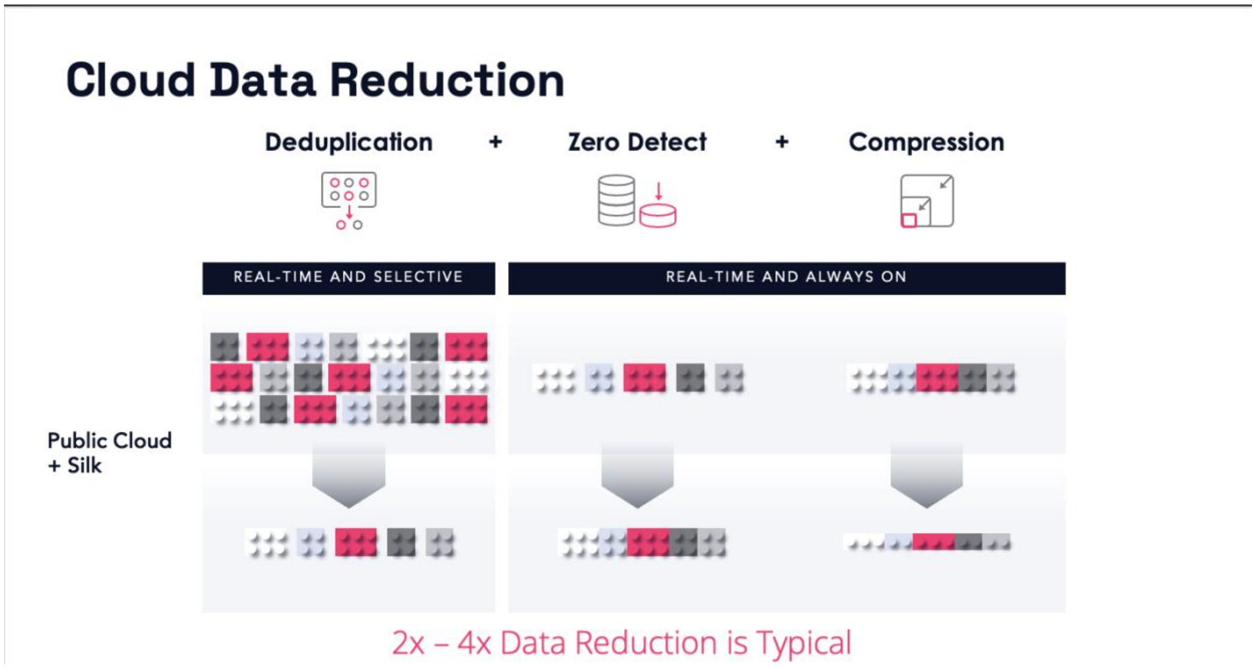
Silk's media layer, also known as an m. node, provides protection with a patented algorithm and double parity. Consistent checkpointing offers data preservation even if there is a zone outage.

Additional performance and storage savings is provided by Silk's proprietary data reduction capabilities.

Scalability with Silk

Scalability is accomplished in two ways with Silk- the first is to achieve performance via the c.nodes. An initial deployment can consist of two c.nodes and scale up to 8 c.nodes. Each c.node can provide additional compute and IO performance to the Silk data pod.

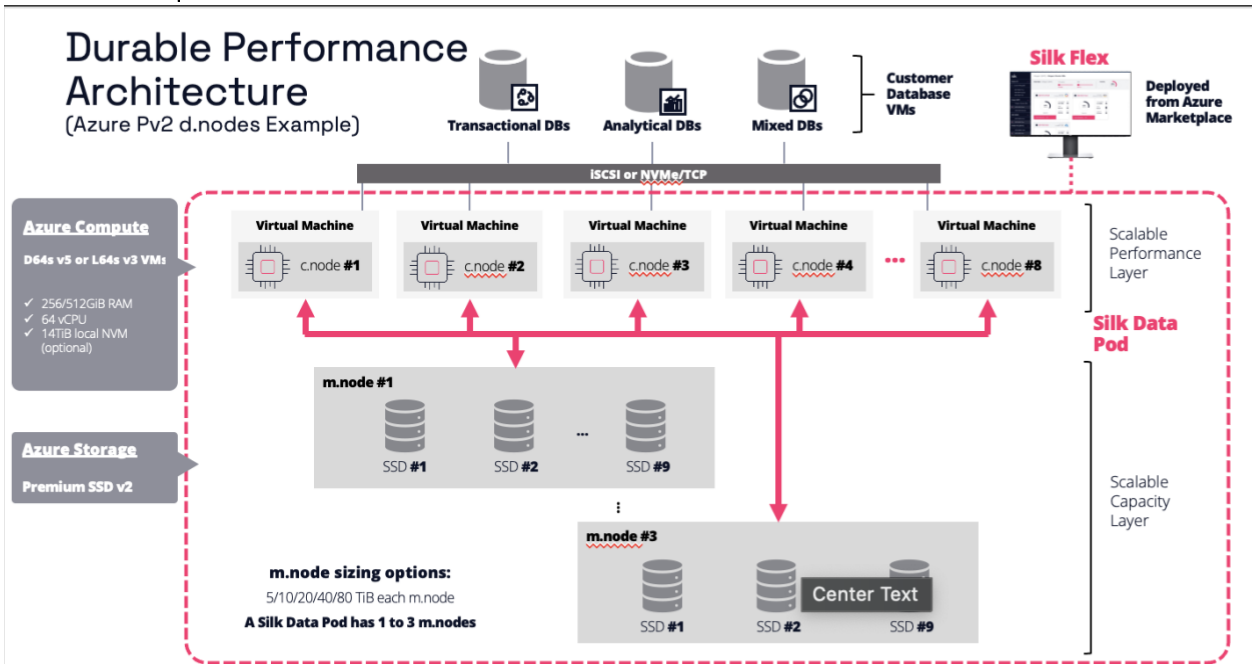
Scaling capacity, which can happen dynamically, involves additional m.nodes, with d.nodes underneath providing resiliency.



The beneficial data reduction features are available no matter what database platform you're on, which cloud you deploy Silk to, or which Silk solution you choose.

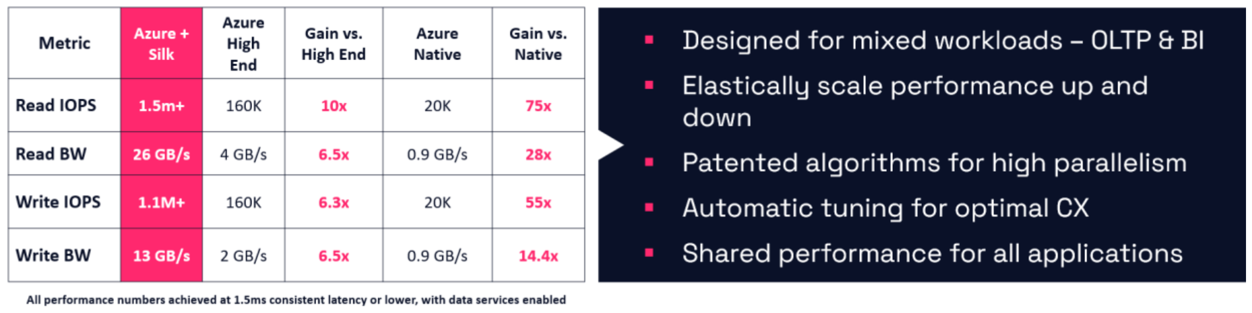
Durable Performance Architecture

The second option in Azure is the Durable Performance Architecture.



This newest option for [Silk](), offers the cost saving and durability of Azure new Premium SSD V2 (PV2), while providing significantly higher IO than native deployments could reach. Additional scalability and resiliency are built into the platform, just as they are with the high-performance architecture, providing the same incredible features at a lower cost. The durable performance solution has the additional feature of read cache which provides an additional performance bump for your read only workloads.

Scaling for performance and scalability is achieved much like it is for the maximum performance architecture, but without the additional compute layer in the m.nodes and taking advantage of the durable performance read cache to continue to scale.



Metric	Azure + Silk	Azure High End	Gain vs. High End	Azure Native	Gain vs. Native
Read IOPS	1.5m+	160K	10x	20K	75x
Read BW	26 GB/s	4 GB/s	6.5x	0.9 GB/s	28x
Write IOPS	1.1M+	160K	6.3x	20K	55x
Write BW	13 GB/s	2 GB/s	6.5x	0.9 GB/s	14.4x

All performance numbers achieved at 1.5ms consistent latency or lower, with data services enabled

- Designed for mixed workloads – OLTP & BI
- Elastically scale performance up and down
- Patented algorithms for high parallelism
- Automatic tuning for optimal CX
- Shared performance for all applications

Advanced Features

One of the Silk's greatest contributions is its value regarding total cost of ownership (TCO). Unlike simple storage, a Silk high IO solution not only meets the IO demands of SQL Server workloads, but it also provides storage savings with its impressive data reduction capabilities. For multi-tier environments, where application and database are tightly coupled, or where numerous monthly refreshes and clones are required, thin provisioning of the entire stack can provide impressive cloud cost savings and valuable time back for database administrators and developers.

Application consistent volume snapshots can relieve IO pressure on VMs to allow for additional sizing decrease, both on compute and on additional SQL Server licensing costs which must license on the vCPU allocated on the VM.

As architecture is designed, it's expected to use Always-on Availability Groups, (AGs) for Disaster Recovery (DR), for secondary replicas for read-only workloads and for even zonal replication. Silk can decrease some demands on licensing and secondary systems by using thin provisioning to create thin provisioned copies in time of need for DR/BC, if SLAs are met. Read-only replicas for short term use can provide zero-footprint storage demands for reporting and testing scenarios.

Summary

Silk on Azure represents a cutting-edge solution for enterprises seeking to optimize their SQL Server workloads, especially those with high IO demands. By leveraging Silk's unique platform, organizations can tap into unparalleled levels of resiliency, ensuring that their data remains accessible and secure even in challenging scenarios. The platform's innate scalability enables businesses to adapt to evolving data needs seamlessly, mitigating the traditional pains of infrastructure expansion. But what truly sets Silk on Azure apart for SQL Server environments is its high IO features. By offering accelerated data throughput and reduced latency, Silk ensures that high IO workloads run smoothly and efficiently. In essence, for businesses looking to elevate their SQL Server performance on Azure, Silk provides a compelling blend of resilience, adaptability, and raw IO prowess.