Maximizing Microsoft SQL Server Performance using Amazon EC2 NVMe Instance Store

July 2020

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

Amazon Elastic Block Store (EBS) provides multiple volume types with various performance and throughput capabilities. Making EBS storage the preferred option for Microsoft SQL Server deployments on AWS. However, there are multiple factors that need to be considered when designing storage configuration for RDMS deployment. This paper presents an approach directed at increasing SQL Server performance through the use of NVMe disks as the write-back cache in front of traditional EBS volumes.

Introduction

Selecting proper configuration for storage subsystem is one of the critical tasks when planning MS SQL Server deployment in the cloud. Every Amazon Elastic Compute Cloud (Amazon EC2) machine instance can be provisioned with Amazon Elastic Block Store (EBS) persistent block storage. EBS volumes are highly available and reliable storage volumes that can be attached to any instance. EBS volumes that are attached to an EC2 instance are exposed as storage volumes that persist independently from the life of the instance.

This paper presents and analyzes a complementary approach directed at increasing SQL Server performance through the use of NVMe disks as the write-back cache in front of traditional EBS volumes. If you are a database administrator concerned about the performance of your SQL Server or a system administrator planning for the expansion of the environment to accommodate ever-increasing load, this paper may provide you with a cost-effective way to improve performance of the disk subsystem for SQL Server, and enable you to improve performance of the existing system or allow for adding more load without reducing performance.

Many of the new Nitro instance types provide instance store using ultra-low latency local NVMe drives physically attached to the host. However, due to temporal nature of the instance store, these NVMe drives were used primarily to host MS SQL Server TempDB. This paper focuses on using local NVMe as a Write-Back Cache in front of an EBS volume, review performance benefits that this approach provides, and considers options to mitigate the temporal nature of local NVMe instance store.

Environment Setup

Window Storage Spaces

Windows Storage Spaces (WSS) is a storage virtualization technology developed by Microsoft and introduced with Windows Server 2012 r2. This technology enables you to virtualize storage by grouping industry-standard disks into storage pools, and then creating virtual disks called storage spaces from the available capacity in the storage pools. Conceptually it is like RAID, implemented in software and is primarily used to create flexible redundant drives. However, one of the important features of this technology, is the ability to configure fast SSD drives as a write-back cache for slower drives, EBS in our case.

For performance testing we will use a mid-range instance, **R5D.4XLARGE**, which is recommended for medium size MS SQL Server databases. This instance comes with 16 vCPUs, 128 GB of RAM, and 2 NVMe drives at 300 GB each. One of these SSD drives will be used to host TempDB, while the fraction of the second drive will be used as the write-back cache as we will perform tests with cache of various sizes. **R5D.4XLARGE** supports sustained EBS throughput 593.75 MB/s (128 KiB I/O) or 18,750 IOPS (16 KiB I/O).

The instance is provisioned with two 4TB EBS drives, one of them is used directly, while the other one configured with the write-back cache on NVMe drive. Each drive provides 12,288 IOPS, which is less than the instance limit, so when the drive is used individually it can deliver its maximum throughput.

After setting up the instance and configuring write-back cache for one of the drives, a preliminary performance testing of the disk subsystem using *CrystalDiskMark* application from the Windows Store was performed. The results of the test are presented in Table 1 below:

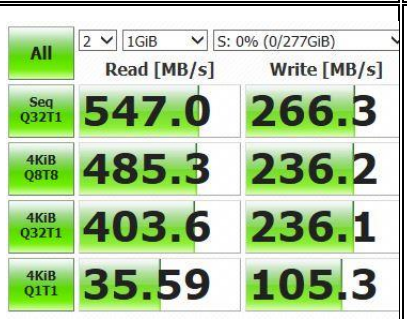
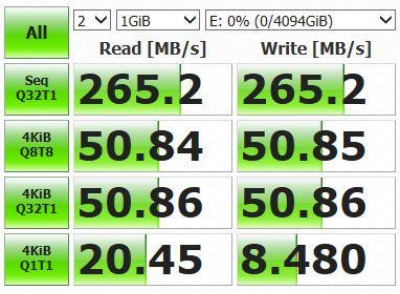
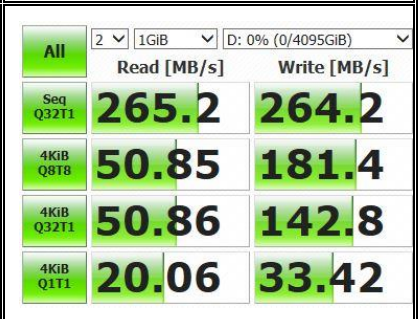
NVMe drive	4-TB EBS drive	4-TB EBS drive WBC on NVMe
		

Table 1 - Performance Comparison of various drives

The read performance of the drive with the write-back cache is the same as the one for the native drive; the same is true for sequential write test as for this instance the NVMe sequential write speed matches the write speed of the 4-TB EBS drive. For the random writes, on the other hand, we clearly see the benefits of the write-back cache.

Performing this test gave us good numbers which indicated that raw disk performance benefits from the write-back cache. Now we need to determine how these numbers would translate into SQL Server performance.

SQL Server Benchmarking Setup

For performance testing we will use the leading benchmarking and load testing software for the most popular databases including SQL Server – HammerDB. We will use the OLTP workload and implement TPC-C benchmark against the databases of various sizes from 2,000 warehouses (HammerDB parameter of the DB size) to 30,000 warehouses, which translates into DB sizes from about 200 GB to 3 TB.

When configuring HammerDB for benchmarking we will use parameter **set allwarehouses** `true`, to increase level of I/O load. For testing, we will use the HammerDB *Autopilot* feature, which enables us to run multiple tests in series. The load level is set by the number of virtual users that HammerDB creates to work against the database.

As there is some deviation in results from test to test with the same load level, each test will be repeated three times and capture TPM (transactions per minute) numbers averaged across these tests. The first test in a series after attaching the new database and restarting SQL Server usually produces significantly lower TPM numbers due to SQL Server initialization (primarily, loading the cache and page buffers), so the results of the first test in Autopilot run are discarded. Thus, the *Active Virtual Users Sequence* for HammerDB *Autopilot* is set to `34 34 34 34 55 55 55 89 89 89 144 144 144 233 233 233` with the understanding that the results of the first benchmark with `34` virtual users will be discarded.

> **Note:** The TPM numbers provided in the rest of this paper **are not indicative of the performance of the SQL Server on AWS** – SQL Server. Underlying Amazon EC2 instances, and EBS subsystems were not configured to obtain maximum performance. The TPM numbers provided are meaningful *only* for comparing relative performance of SQL Server on one particular instance type with the specific workload running against EBS volume configure with or without write-back cache.

SQL Server Performance Testing

We will cover SQL Server single instance as well as SQL Server in Always-On Availability Group as our test cases, and then proceed with evaluation write-back caching combined with RAID 0 across EBS volumes.

SQL Server Single Instance

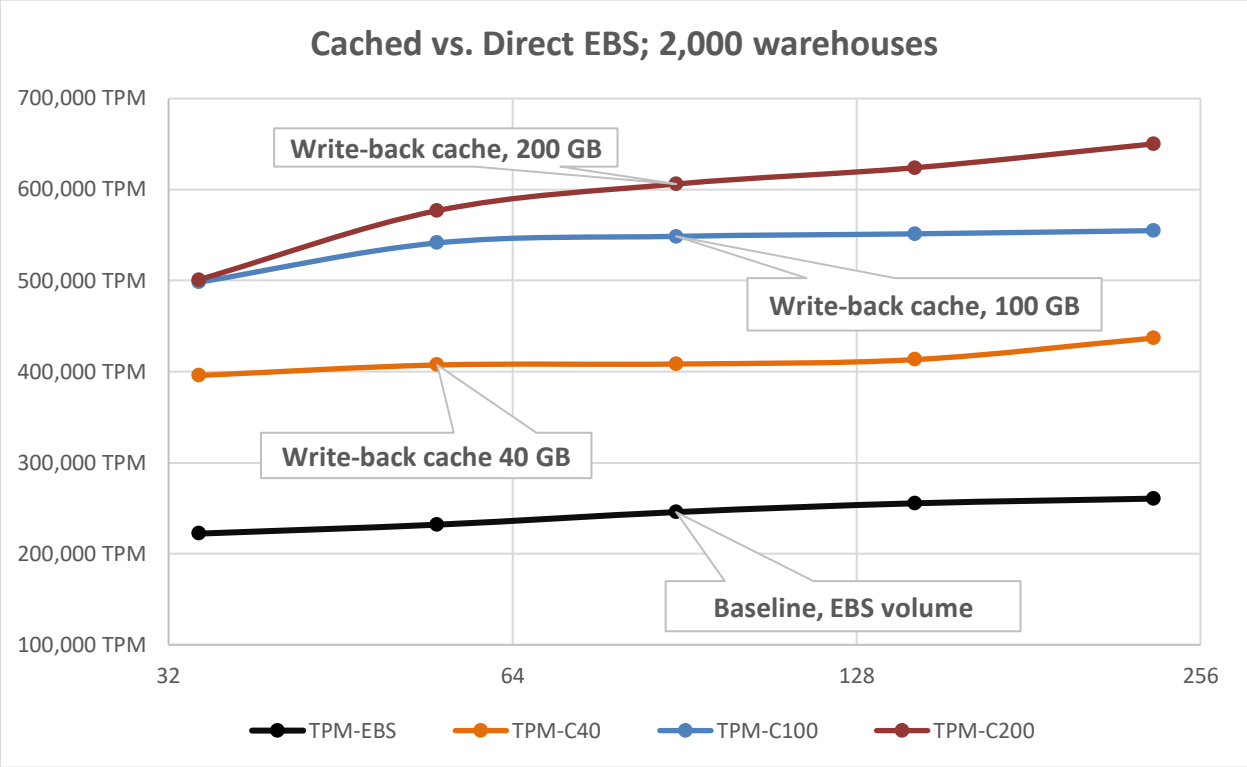


Figure 1 - Benchmarking results for 2,000 warehouses database

Figure 1 above represents the benchmarking results for 2,000 warehouses databases placed on EBS drive versus the same EBS drive with the write-back cache of different sizes. Horizontal axis captures the number of virtual users at logarithmic scale, while the vertical axis represents the values of achieved TPM.

If for each level of workload (number of virtual users) divide the value of the TPM achieved with the respective write-back cache to the baseline TPM value for the same load and then average these numbers across all workloads, we will get the coefficient of performance improvement (Cpi) provided by the write-back cache. If we plot Cpi against the size of the write-back cache, we will get the chart presented on the Figure 2 below.

The write-back cache of 40 GB provides average performance improvement of about **70%**, which grows to **120%** for 100 GB cache, and, finally, to **140%** for 200 GB cache. For this relatively small database going from 100 GB cache to 200 GB cache provides marginal improvement in performance, especially at the lower end of the workload – at lower level of load the 200 GB cache performs the same as the 100GB cache as at this level of load cache is not fully utilized.

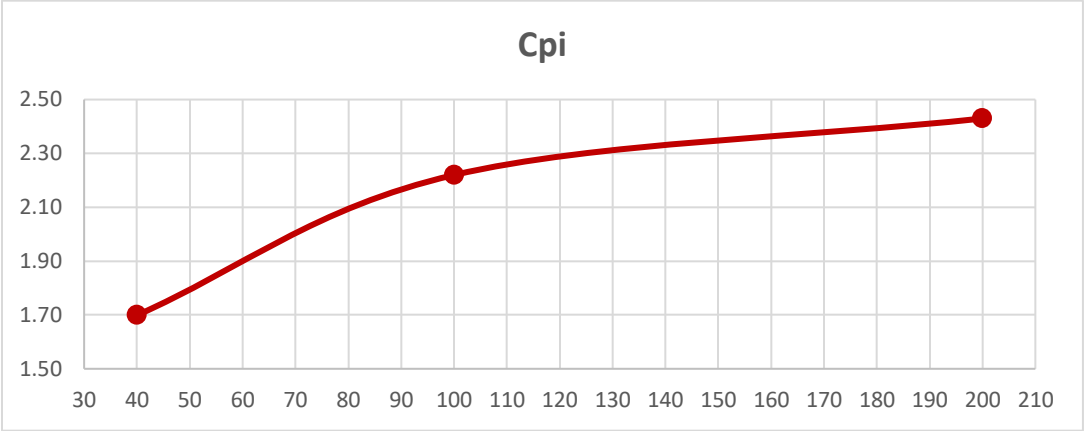


Figure 2 - Coefficient of performance improvement

Now let's see how using the write-back cache affects performance of a larger database, specifically, the one based upon 8,000 warehouses with the size of about 800 GB. The benchmarking results for this database is presented below in Figure 3. The 40 GB cache provides average performance improvement of about **50%**, but the larger cache sizes of 100 GB and 200 GB result in much more impressive performance improvement of about **140%** and **170%** respectively.

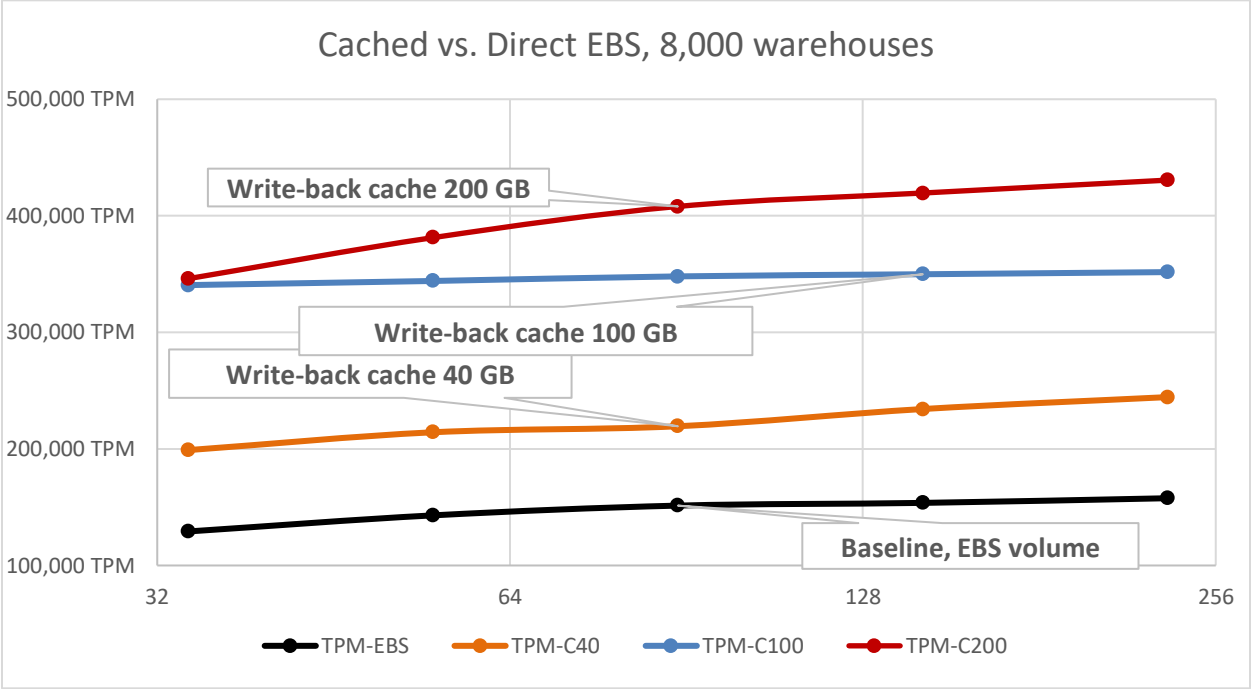


Figure 3 - Benchmarking results for 8,000 warehouses database

It appears from these results that larger databases with OLTP workload would benefit from a larger cache at high level of load. Again, at a low level of workload the difference in performance between 100 GB and 200 GB cache is minimal, but as the level of a workload grows, a larger cache provides meaningful benefits.

The next step is to test performance benefits of write-back cache for really large database of 30,000 warehouses, which translates into the database size of about 3 TB. Typically for the OLTP database of this size a larger instance is recommended than the **R5D.4XLARGE**, which we use for this testing. However, one of the goals of this test is to demonstrate that even the mid-size instance could provide reasonable performance for a quite large database if we use write-back cache in front of the EBS volume. The NVMe Instance Store is not subject to the instance-level limit for EBS throughput and can cope with extensive level of write requests, primarily to the log file, to allow transactions commit without delay and then gradually offload changes to the underlying EBS volume.

However, for benchmarking performance against this database we use a cache size of 100 GB and 200 GB as, judging from the results for 8,000 warehouses database, the 40 GB cache may not provide meaningful improvement in performance for the database of this size. The benchmarking results for the 30,000 warehouses database is presented below in Figure 4.

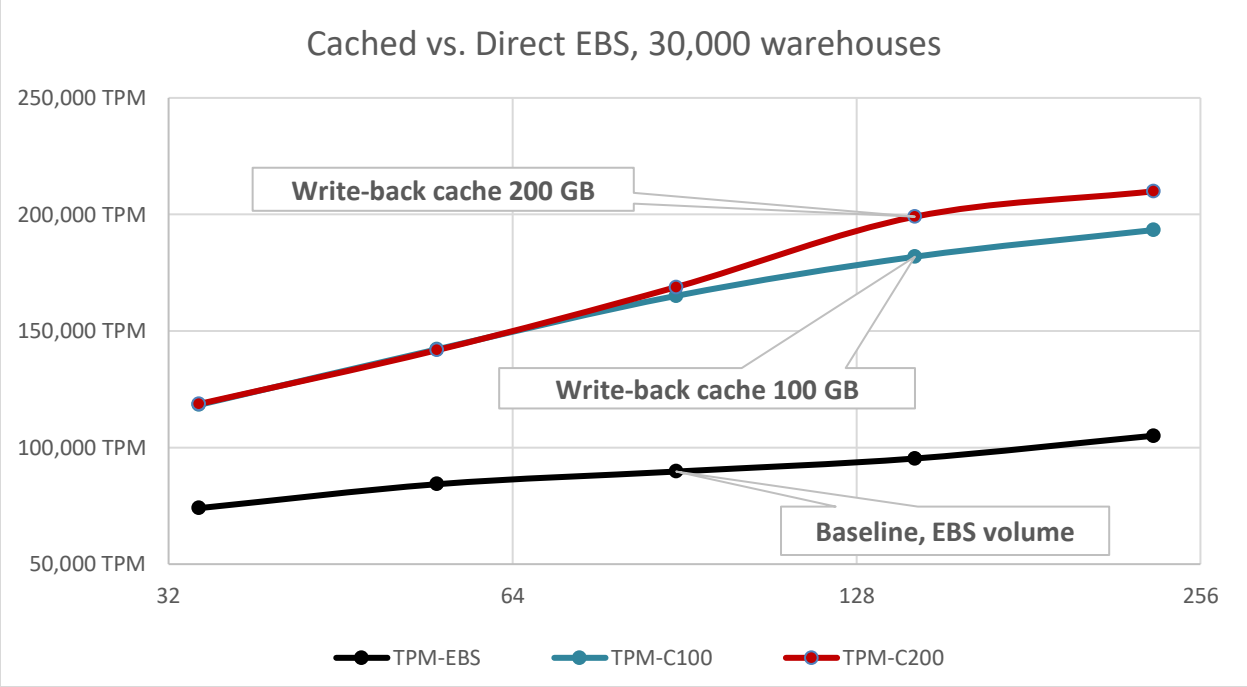


Figure 4 - Benchmarking results for 30,000 warehouses database

The 100 GB cache provides average performance improvement of about **80%** and the 200 GB results in performance improvement of about **85%**. Again, at low level of workload the difference in performance between 100 GB and 200 GB cache is minimal, but as the level of workload grows, larger cache provides meaningful benefit.

Always-On Availability Group

As previously mentioned, if the instance is stopped for reasons such as a hardware upgrade or component failure, the NVMe storage is lost. This is a shortcoming of the WSS as it doesn't recognize that in this particular case, the NVMe storage is used as the cache in front of the persistent EBS storage, so loss of the SSD component should be recoverable. However, this is not the case, and loss of the cache disk results in unrecoverable failure of the whole virtual volume. For a test/development system this may not be that critical, the volume can be recreated, and the database restored from the backup. This approach may not work for production systems due to the substantial downtime required to restore the operational status of the database.

However, most of the production systems already use a high-availability share-nothing solution like SQL Server Always-On Availability Group. In this case the loss of one instance would not affect the availability of the system and downtime of a failed instance required to recreate the volume and restore the database would not result in the system failure, but in temporary reduction in high availability.

Thus, it becomes important to evaluate effect of the write-back cache on the performance of SQL Server configured with Always-On Availability Group with synchronous replication, which would guarantee seamless failover without the data loss in case of failure of primary instance.

For this evaluation we will limit benchmarking to the databases of 8,000 and 30,000 warehouses with the write-back cache of 200 GB. The benchmarking results for 8,000 warehouses database in Always-On Availability Group configuration with synchronous commit is presented below in Figure 5. For comparison, the results for the same database in single-instance configuration, discussed in the previous section, and presented as dotted lines on the same chart. As compared with the single-instance results, providing write-back cache of 200 GB results in slightly larger performance improvement of about **175%** for Always-On Availability Group configuration.

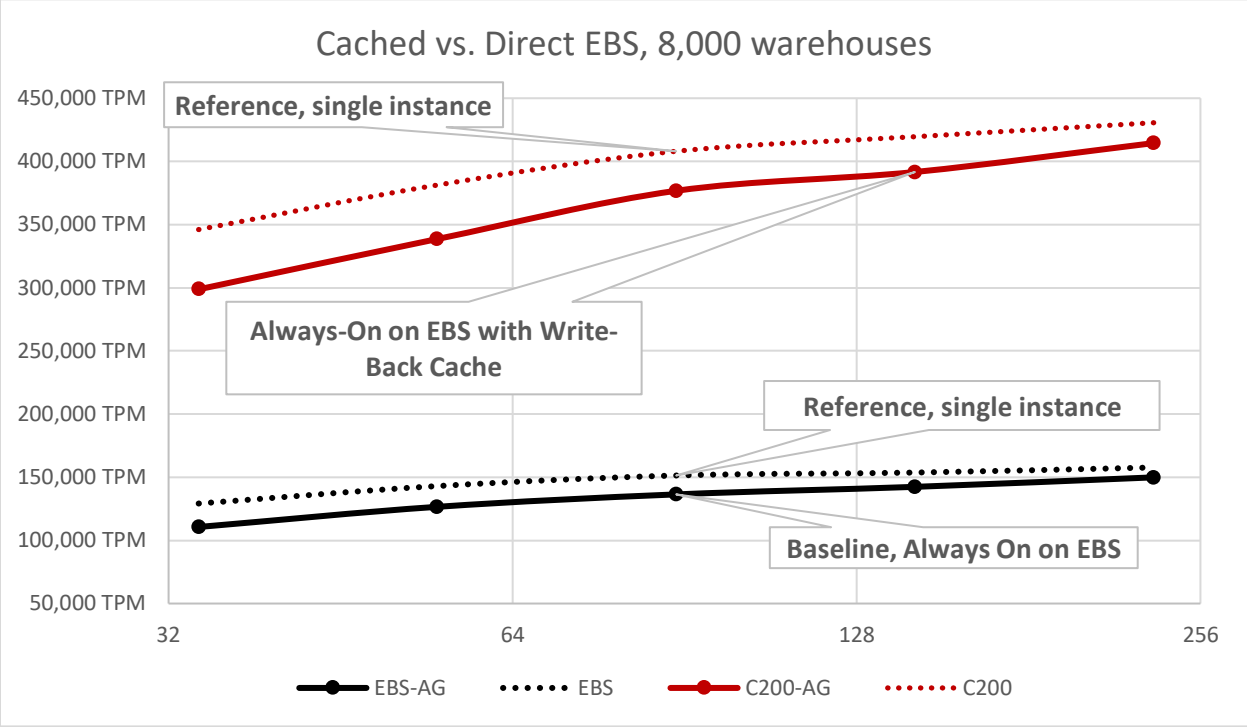


Figure 5 - Benchmarking results for 8,000 warehouses database in Always-On Availability Group with synchronous commit configuration.

Similarly, Figure 6 below presents benchmarking results for the 30,000 warehouses database in Always-On Availability group configuration. Providing 200 GB cache results in about **100%** performance improvement as compared to the same configuration hosted just on EBS volume. This even exceeds the 85% performance improvement that we saw for the same database in a single-instance configuration.

These benchmarks confirm that providing a write-back cache for the EBS volume hosting the database results in significant performance improvement as compared to the same database hosted on EBS volume without cache. If we compare results presented in Figure 1 above and Figure 6 below, we may see that providing 200 GB write-back cache for a 3 TB database in Always-On Availability Group configuration with synchronous replication allows to reach the same level of performance that we achieve for a database 15 times smaller in a single-instance configuration.

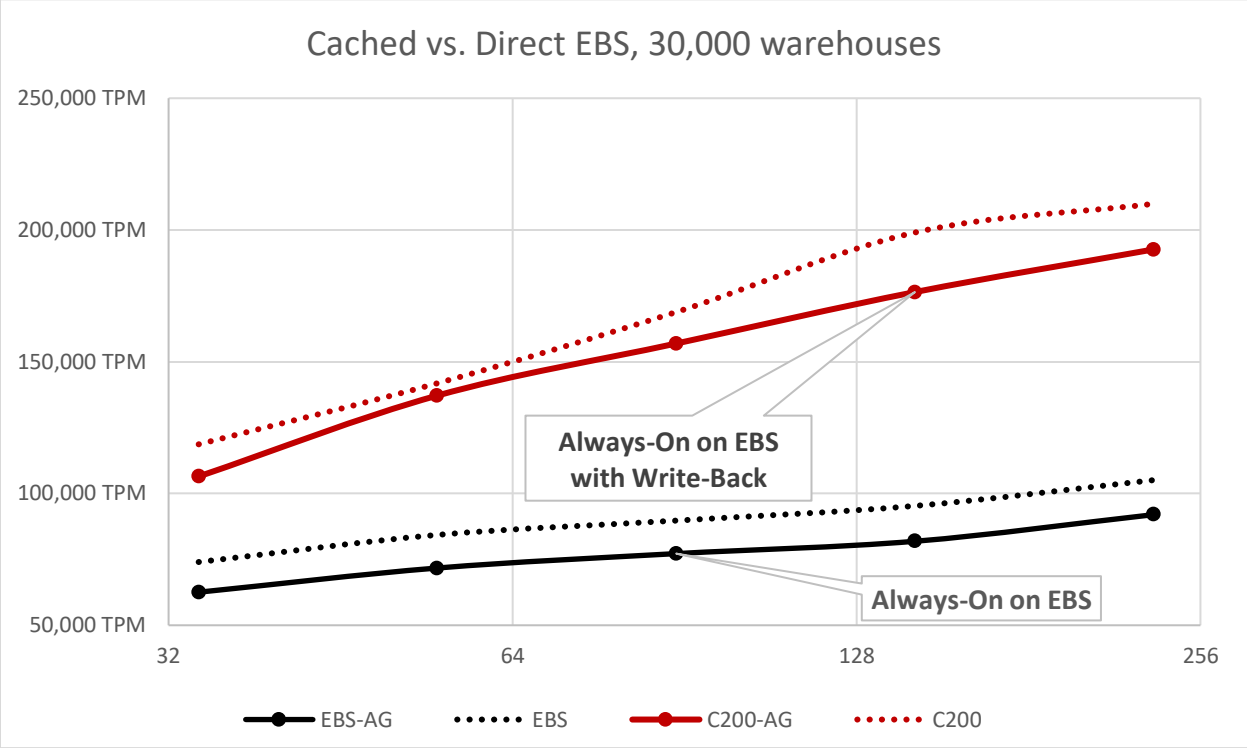


Figure 6 - Benchmarking results for 30,000 warehouses database in Always-On Availability Group with synchronous commit configuration.

Write-back cache and EBS striping

One of the recommended approaches to increase performance of the underlying storage is to use RAID-0 disk striping. To test this, we will compare the performance of a 4 TB drive and a virtual drive of the same size comprised of 4 1-TB drives in RAID-0 striping configuration.

As shown below from the data in Table 2, the RAID-0 striping produces a significantly higher sequential read and write performance. Now it is time to test how striping will affect SQL Server performance and whether providing a write-back NVMe cache in front of a striped volume will improve performance. For this test we will use an **R5D.12XLARGE** instance and ***30,000*** warehouses ***HammerDB*** database. There will be two series of tests; the first test will be against the database located on a striped volume, and the second test will be against the database located on the same volume but with the 200 GB write-back cache in front of it.

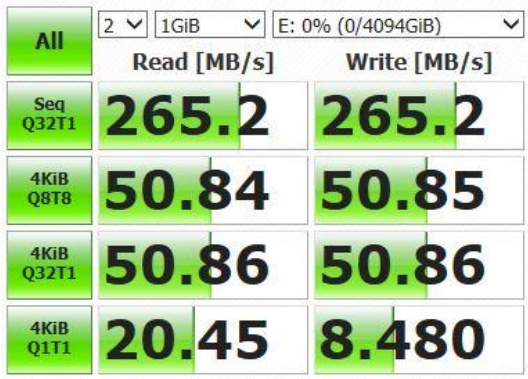
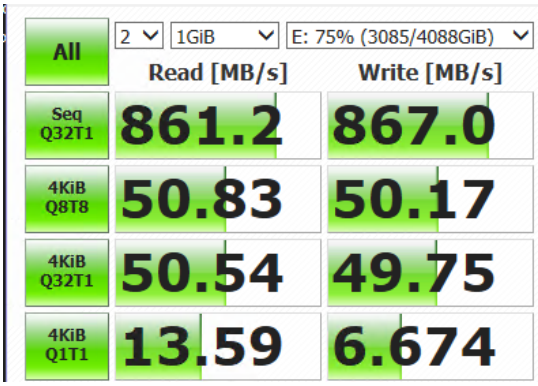
Single 4-TB volume	4 1-TB volumes in RAID-0 configuration
	

Table 2 - Single 4-TB volume vs. 4 1-TB volumes in RAID-0 configuration

As shown in the previous tests, for each set of virtual users three benchmarks will be captured and results averaged to obtain more consistent results. Also, considering the larger instance and faster storage, we will extend the HammerDB Autopilot sequence to include test with **377** virtual users so that the whole sequence will look like `34 34 34 34 55 55 55 89 89 89 144 144 144 233 233 233 377 377 377`. The benchmarking results are presented below in Figure 7.

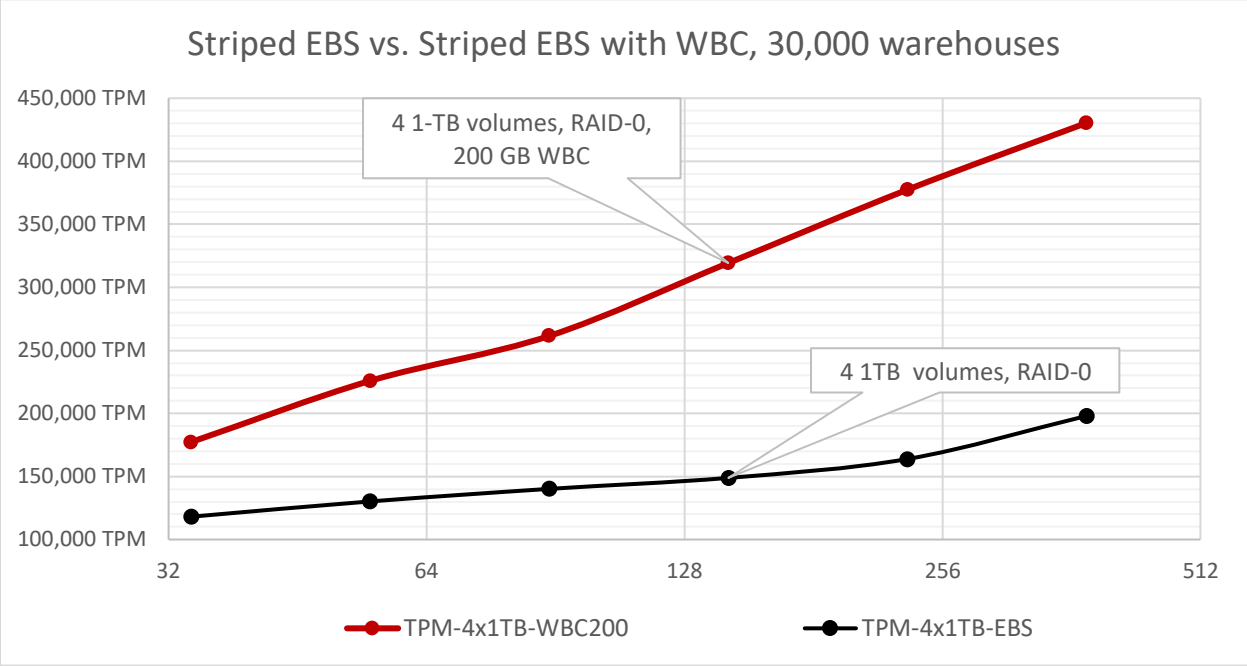


Figure 7 - Benchmarking results for 30,000 warehouses database on striped volume with and without write-back cache.

As shown in the chart, a modest 200 GB NVMe cache in front of a 3 TB database on already optimized through striping EBS-based volume almost doubles performance of SQL Server. The Cpi starts at about **1.5** (a 50% performance improvement) at low levels of load and then exceeds **2.0** (a 100% improvement) for larger loads. The average Cpi for all data points on the chart is **1.95**.Figure 7 - Benchmarking results for 30,000 warehouses database on striped volume with and without write-back cache.

Automating configuration of write-back cache and related operations using PowerShell scripts

There are multiple ways to organize your storage subsystem for SQL Server in Amazon EC2, using native features in combination with WSS. It would be difficult and impossible to develop a script that would cover the multitude of available EC2 instance configurations and EBS storage configurations. In this section, we will present and discuss a PowerShell initialization script and corresponding shutdown script for a configuration which includes the following:

- An instance with at least two NVMe volumes; the first one will be used individually to create a drive to host TempDB, and a fraction of the second one will be used as a write-back cache.
- One or more large EBS volume to host SQL Server database files; if more than one volume available for database, they will be combined into a RAID-0.
- One small EBS volume will be used as swap space for NVMe-based write-back cache.

The target storage configuration is presented below in Figure 8. The Storage Pool SSD contains one of the two NVMe drives. This drive is the base for the SSD virtual disk, which is attached to the host as drive *S:*\. The SQL Server is configured to put *TempDB* on this drive. The storage pool, virtual disk, and S:\ drive is re-created on system start-up. This is not an issue because the SQL Server creates a new *TempDB* if the files aren't found. The only requirement is that SQL Server should be configured for a *Delayed* or *Manual* start so that the drive is created before SQL Server needs it.

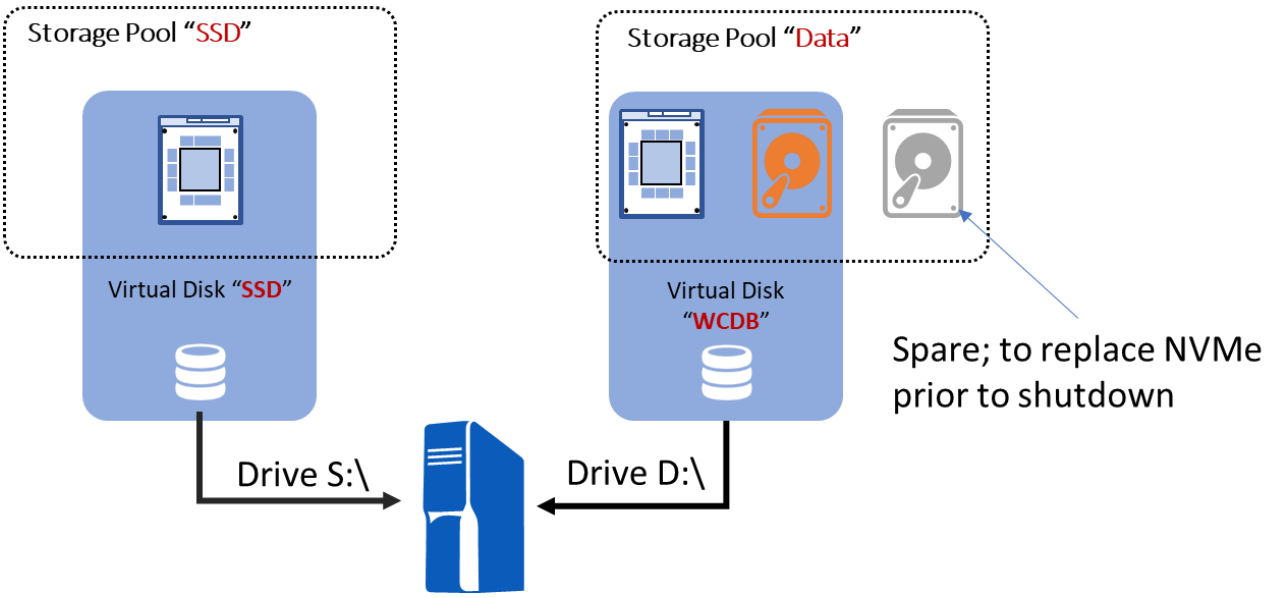


Figure 8 - Storage Configuration

Storage pool **DATA** includes the second NVMe drive, the large EBS volume(s) for SQL Server data, and an EBS volume sufficient to accommodate a configured write-back cache, which is re-allocated to this volume prior to scheduled shutdown to preserve *health* of the cached drive. For more information see, Shutdown script. Under normal operational condition, the virtual disk **WCDB** contains the EBS volume(s) and the

second NVMe drive configured as the write-back cache. This virtual drive is attached to the host as drive *D:*\. The SQL Server is configured to place the database and log files for the HammerDB database used for performance testing on this drive.

The scripts were tested against instances of various sizes in **R5D** family with one or more EBS volumes for hosting the database. The *initialization* script will perform the following actions:

1) Creates an **SSD** storage pool using one of the NVMe disks, an **SSD** volume in this pool, and creates and formats the *S:*\ drive on this volume. The *S:*\ drive will be used for TempDB, a recommended approach for instances with NVMe storage as SQL Server will re-create TempDB upon start-up if it is not present.
2) Checks for the presence and status of the **DATA** storage pool. If the pool is in an **UNHEALTHY** state (this could be the result of unexpected shutdown/failure of the instance), it drops the pool, and releases its resources.
3) Checks for the presence of **DATA** storage pool. If pool does not exist (initial start of the instance or start after unexpected shutdown/failure with the pool deleted on previous step), it will create a **DATA** storage pool using the second of NVMe drive and available EBS volumes, a **WCDB** volume using one or more large EBS volumes with the write-back cache on NVMe, and will mark the smaller of the EBS volumes as **RETIRED** to reserve it in case cache needs to be moved to EBS prior to scheduled shutdown. It also creates and formats the *D:*\ drive on a **WCDB** volume. The *D:*\ drive is configured with write-back cache on NVMe and will be used for SQL Server user database(s).
4) If the **DATA** storage pool exists, it checks whether the write-back cache is on NVMe. If cache *is not on SSD drive* (could be the result of re-allocating cache to EBS volume by the ***shutdown*** script to preserve integrity of the **WCDB** volume ahead of scheduled ***stopping*** of the instance), it adds the available NVMe disk to the **DATA** pool, and re-allocates write-back cache to the SSD to restore drive *D:*\ performance.
5) Verifies that the **WCDB** volume is attached to the instance. In some cases, it may get detached at shutdown/startup.
6) Captures the current state of the storage configuration on the instance to the execution log.

The initialization script should be placed into the User Data section of the respective EC2 instance and configured to run on every start by using the `<persist>` tag:

```
<powershell>
$CacheSize = 100    # Cache Size in GB
… The rest of the script …
</powershell>
```

```
<persist>true</persist>
```

Note: The first line of the initialization script defines and sets the value of the variable that controls the size of the allocated write-back cache in gigabytes. This value should not exceed the amount of space available on NVMe of the respective instance. The initialization script execution log generated by the `Write-Host` cmdlets in the script can be found in the `UserdataExecution.log` file located at `C:\ProgramData\Amazon\EC2-Windows\Launch\Log\` for Windows Server 2016 and above.

Shown below is an example log generated at the initial instance startup:

```
2020/02/09 20:19:06Z: Userdata execution begins
2020/02/09 20:19:06Z: <persist> tag was provided: true
2020/02/09 20:19:06Z: Running userdata on every boot
2020/02/09 20:19:07Z: <powershell> tag was provided.. running
powershell content
2020/02/09 20:19:30Z: Message: The output from user scripts:
SSD storage pool does not exist - Creating
SSD storage pool created
Data storage pool does not exist - Creating
Data storage pool does not exist - Creating
Storage configuration

FriendlyName          HealthStatus OperationalStatus Size
------------          ------------ ----------------- ----
NVMe Amazon Elastic B Healthy      Online              100
WCDB                  Healthy      Online              999
SSD                   Healthy      Online              277

DriveLetter FriendlyName HealthStatus OpStatus SizeRemaining
Size
----------- ------------ ------------ -------- -------------      -
---
C           System       Healthy      OK            62.27 GB    100
GB
D           WCDB         Healthy      OK           998.84 GB 998.98
GB
S           SSD          Healthy      OK           276.88 GB 276.98
GB
```

```
2020/02/09 20:19:30Z: Userdata execution done
```

The shutdown script could be started manually or by some other means, such as using the *On-Shutdown feature* of the local policy, ahead of the scheduled stopping of the instance. The script validates necessary conditions, then re-allocates write-back cache from the NVMe drive to the spare small EBS volume that was added to the **DATA** pool by initialization script in the `retired` state. The size of this EBS volume should be sufficient to hold the allocated write-back cache. The script also stops SQL Server service and SQL Server agent. Stopping the SQL Server is not required, however, if it keeps running, the performance of SQL Server would be significantly impaired during cache reallocation.

> **Note:** Swapping a drive for write-back cache can take a long time. Theoretically, if WSS realized that the drive being swapped is used as cache, it would be sufficient to just flush the cache onto the underlying volume and then swap the drives. Unfortunately, this is not the case so the WSS actually copying everything from the write-back cache disk being retired to the new one. For large cache sizes (100 GB, 200 GB) this operation may take significant time.

In the tests on the **R5D.4XLARGE,** transferring a cache to a 300 GB EBS took about 4 seconds per GB of allocated cache. Using a larger instances or faster drives, this time can be reduced.

Conclusion

Using the write-back cache feature of the Windows Storage Spaces provides a new way to maximize SQL Server performance on AWS, which can be combined with the approaches outlined in the Maximizing Microsoft SQL Server Performance with Amazon EBS blog post. Significant performance improvements provided by utilizing write-back cache on NVMe enables you to:

- Achieve a ***higher SQL Server performance*** without migrating to more powerful instance type and/or using EBS volumes with provisioned IOPS.
- Achieve ***cost savings*** by downgrading installation to a smaller instance and/or removing provisioned IOPS from the EBS volumes without sacrificing performance.

Due to the temporal (non-persistent) nature of the NVMes, this approach to improve SQL Server performance, for all practical cases except for some development

databases, would require deployment of Always-On Availability Group(s) with synchronous replication in either Standard or Enterprise edition of SQL Server. However, this requirement does not look excessive as most production workloads are anyway deployed in high-availability configuration.

The temporal nature of NVMes can be ***mitigated for scheduled stopping*** of the instance by re-allocating cache to a spare EBS volume and then upon start-up of the instance re-allocating cache back to new NVMe.

Appendix A: Initialization/Recovery and Shutdown Scripts

The following scripts were developed to address the configuration discussed in this paper. If your configuration is different, you can use these scripts as a sample template to assist you in developing scripts to address your specific needs.

Initialization script

```
$CacheSize = 10    # Cache Size in GB
Write-Host " "     # Create new line in the log
# ------------------------------------------------------------------
------------
# Check for presence of SSD data pool to host SQL Server TempDB
# ------------------------------------------------------------------
------------
if ( !(Get-StoragePool | ? FriendlyName -EQ SSD))
    {
    Write-Host "SSD storage pool does not exist - Creating"
    # ------------------------------------------------------------
------------
    # Create "SSD" storage pool and respective drive to host SQL
Server TempDB
    # ------------------------------------------------------------
------------
    New-StoragePool -StorageSubSystemId (Get-
StorageSubSystem).UniqueId `
            -FriendlyName SSD `
            -PhysicalDisks ( Get-PhysicalDisk  -CanPool $true `
                                               -FriendlyName 'NVMe
Amazon EC2 NVMe' |
```

```
                                                    Select -First 1 ) |
Out-Null
    # ------------------------------------------------------------
------------
    # Create virtual disk "SSD" on pool "SSD" to host SQL Server
TempDB
    # ------------------------------------------------------------
------------
    Get-StoragePool SSD | New-VirtualDisk  -FriendlyName SSD `
                                           -ProvisioningType Fixed
`
                                           -UseMaximumSize `
                                           -ResiliencySettingName
Simple | Out-Null
    # ------------------------------------------------------------
------------
    # Configure volume "S" on virtual disk "SSD" to host SQL Server
TempDB
    # ------------------------------------------------------------
------------
    Get-VirtualDisk SSD | Get-Disk | Set-Disk -IsReadOnly 0
    Get-VirtualDisk SSD | Get-Disk | Set-Disk -IsOffline 0
    Get-VirtualDisk SSD | Get-Disk | Initialize-Disk -
PartitionStyle GPT -Confirm:$false
    Stop-Service -Name ShellHWDetection   # Required to avoid pop-
up dialog
    Get-VirtualDisk SSD | Get-Disk | New-Partition -Alignment
1024KB `
                                                   -DriveLetter "S"
`
                                                   -UseMaximumSize
| Out-Null
    Format-Volume -DriveLetter "S" -FileSystem NTFS `
                  -NewFileSystemLabel SSD `
                  -AllocationUnitSize 64KB -Confirm:$false | Out-
Null
    Start-Service -Name ShellHWDetection   # Restore service
    # ------------------------------------------------------------
------------
    Write-Host "SSD storage pool created"
    }

# ------------------------------------------------------------------
------------
```

```
# Check for presence and status of the Data pool which hosts DB
# ------------------------------------------------------------------
------------
if ( (Get-StoragePool | ? FriendlyName -EQ Data) `
    -AND `
    (Get-StoragePool | ? FriendlyName -EQ Data).HealthStatus -NE
"Healthy")
    {
    Write-Host "Data storage pool exists but not healthy -
Deleting"
    # ------------------------------------------------------------
------------
    # Remove Data Storage pool - the DB will be UNAVALABLE!!!
    # ------------------------------------------------------------
------------
    Get-StoragePool Data | Get-PhysicalDisk | ? FriendlyName -EQ
NVME |
        Set-PhysicalDisk -Usage Retired -NewFriendlyName
RetiredNVME
    #------------------------------------------------------------
---
    Remove-VirtualDisk -FriendlyName WCDB -Confirm:$false
    #------------------------------------------------------------
---
    Remove-StoragePool -FriendlyName Data -Confirm:$false
    #------------------------------------------------------------
---
    Write-Host "Unhealthy Data storage pool Deleted"
    }

# ------------------------------------------------------------------
------------
# If it does not exist, create Data pool which hosts DB - could be
on initial
# system configuration or may follow deletion of the pool due to
unexpected
# loss of the efemeral drive (server was stopped without
reconfiguring pool).
# In the latter case the DB(s) need to be restored from the backup
or by some
# other means.
# ------------------------------------------------------------------
------------
if ( !(Get-StoragePool | ? FriendlyName -EQ Data) )
```

```
    {
    Write-Host "Data storage pool does not exist - Creating"
    # ---------------------------------------------------------------
------------
    # Create "Data" storage pool and respective drive to host SQL
Server DB(s)
    # on the SSD-writecached drive
    # ---------------------------------------------------------------
------------
    New-StoragePool -StorageSubSystemId (Get-
StorageSubSystem).UniqueId `
            -FriendlyName Data `
            -PhysicalDisks ( Get-PhysicalDisk  -CanPool $true )  |
Out-Null
    # ---------------------------------------------------------------
------------
    # Set FriendlyName for the drive that will be used as Swap
Space for SSD Cache
    # NOTE: For swap we select the smallest of the EBS drives
    # ---------------------------------------------------------------
------------
    Get-StoragePool Data | Get-PhysicalDisk | ? FriendlyName -EQ
'NVMe Amazon Elastic B' |
        Sort Size | Select -First 1 |
        Set-PhysicalDisk -MediaType SSD -Usage Retired -
NewFriendlyName Swap
    # ---------------------------------------------------------------
------------
    # Set MediaType and FriendlyName for the drive(s) that will be
used for HDD Tier
    # ---------------------------------------------------------------
------------
    Get-StoragePool Data | Get-PhysicalDisk | ? FriendlyName -EQ
'NVMe Amazon Elastic B' |
        Set-PhysicalDisk -MediaType HDD -Usage AutoSelect -
NewFriendlyName Data
    # ---------------------------------------------------------------
------------
    # Set FriendlyName and Usage for the NVMe drive(s) so they
could be used as Cache
    # ---------------------------------------------------------------
------------
    Get-StoragePool Data | Get-PhysicalDisk | ? FriendlyName -EQ
'NVMe Amazon EC2 NVMe' |
```

```
        Set-PhysicalDisk -MediaType SSD -Usage Journal -
NewFriendlyName NVME

    # ---------------------------------------------------------------
------------
    # Create virtual disk "WCDB" on pool "Data" to host SQL Server
database
    # ---------------------------------------------------------------
------------
    Get-StoragePool Data | New-VirtualDisk  -FriendlyName WCDB `
                                            -ProvisioningType Fixed
`
                                            -UseMaximumSize `
                                            -ResiliencySettingName
Simple `
                                            -WriteCacheSize
($CacheSize * 1073741824)  |
        Out-Null
    # ---------------------------------------------------------------
------------
    # Configure volume "D"□ on Write-Cached virtual disk
    # ---------------------------------------------------------------
------------
    Get-VirtualDisk WCDB | Get-Disk | Set-Disk -IsReadOnly 0
    Get-VirtualDisk WCDB | Get-Disk | Set-Disk -IsOffline 0
    Get-VirtualDisk WCDB | Get-Disk | Initialize-Disk -
PartitionStyle GPT -Confirm:$false
    Stop-Service -Name ShellHWDetection   # Required to avoid pop-
up dialog
    Get-VirtualDisk WCDB | Get-Disk |
                    New-Partition -Alignment 1024KB -DriveLetter
"D" -UseMaximumSize |
                    Out-Null
    Format-Volume -DriveLetter "D" -FileSystem NTFS `
                  -NewFileSystemLabel WCDB `
                  -AllocationUnitSize 64KB -Confirm:$false | Out-
Null
    Start-Service -Name ShellHWDetection   # Restore service
    # ---------------------------------------------------------------
------------
    Write-Host "Data storage pool does not exist - Creating"
    }
```

```
# ------------------------------------------------------------------
------------
# If the Data pool was prepared for instance shutdown and loss of
efemeral
# storage, we may recover optimal (WriteBackCache on ephemeral SSD)
Data pool
# configuration.
# Please note that depending on the size of the cache, this
operation may take
# long time!
# ------------------------------------------------------------------
------------
if (     (Get-StoragePool | ? FriendlyName -EQ Data) `
    -AND `
         (Get-StoragePool | ? FriendlyName -EQ Data).HealthStatus -
EQ "Healthy" `
    -AND `
         (Get-VirtualDisk | ? FriendlyName -EQ WCDB) `
    -AND `
         (Get-PhysicalDisk | ? FriendlyName -EQ Swap).Usage -EQ
"Journal" `
    -AND `
         (Get-PhysicalDisk | ? CanPool -EQ $true | ? FriendlyName -
EQ "NVMe Amazon EC2 NVMe")
    )
    {
    Write-Host "Recovery conditions met - recovering WCDB drive"
    #------------------------------------------------------------------
------------------
    $Swap = Get-PhysicalDisk | ? FriendlyName -EQ Swap
    $NVME = Get-PhysicalDisk | ? CanPool -EQ $true | ? FriendlyName
-EQ "NVMe Amazon EC2 NVMe"
    #------------------------------------------------------------------
------------------
    # Add new NVMe drive to Storage Pool
    Add-PhysicalDisk -PhysicalDisks $NVME -StoragePoolFriendlyName
Data
    # Set attributes of the new NVMe drive
    Set-PhysicalDisk -InputObject $NVME -Usage Journal -
NewFriendlyName NVME
    # Set attributes of the Swap drive
    Set-PhysicalDisk -InputObject $Swap -Usage Retired
    #------------------------------------------------------------------
------------------
```

```
    # Reallocation cache to NVMe
    Repair-VirtualDisk -FriendlyName WCDB
    #-------------------------------------------------------------
-------------------
    Write-Host "Recovery of WCDB drive completed successfully"
    }

# -------------------------------------------------------------
------------
# It may happen that virtual disk WCDB is left in the "Detached"
state upon
# recovery - we need to re-attach it to the host
# -------------------------------------------------------------
------------
if ( "Detached" -EQ (Get-VirtualDisk | ? FriendlyName -EQ
WCDB).OperationalStatus )
    {
    Write-Host "WCDB Virtual Disk is <Detached> - attaching"
    Connect-VirtualDisk -FriendlyName WCDB
    }
# -------------------------------------------------------------
------------
# Current status capture
# -------------------------------------------------------------
------------
Write-Host "Storage configuration"
Get-Disk | FT FriendlyName, HealthStatus, OperationalStatus,
@{n="Size";e={[math]::Round($_.Size/1GB,2)}}
Get-Volume | ? DriveType -EQ Fixed
```

Shutdown script

```
if (!(Get-StoragePool | ? FriendlyName -EQ Data))
    {
    Write-Host "Target storage pool not found - exiting"
    exit
    }

$NVME = Get-PhysicalDisk | ? FriendlyName -EQ NVME
if (!$NVME)
    {
    Write-Host "Cache (NVME) drive not found - exiting"
    exit
```

```
    }

$Swap = Get-PhysicalDisk | ? FriendlyName -EQ Swap
if (!$Swap)
    {
    Write-Host "Swap drive not found - exiting"
    exit
    }

if ($Swap.Usage -NE "Retired")
    {
    Write-Host "Swap drive is not in RETIRED usage state - exiting"
    exit
    }

$WCDB = Get-VirtualDisk | ? FriendlyName -EQ WCDB
if (!$WCDB)
    {
    Write-Host "Taqrget virtual disk WCDB not found - exiting"
    exit
    }

Write-Host "Found new Swap drive - proceeding with swapping cache
drive for EBS"
#-------------------------------------------------------------
Stop-Service -Name SQLSERVERAGENT
Stop-Service -Name MSSQLSERVER
#-------------------------------------------------------------

# Change Swap disk usage to AutoSelect to enable media swap
Set-PhysicalDisk  -FriendlyName Swap -MediaType SSD -Usage Journal

# Change cache drive NVME usage to Retired to enable removal
Set-PhysicalDisk  -FriendlyName NVME -Usage Retired

# Reallocation....
Repair-VirtualDisk -FriendlyName WCDB

# Delete NVMe drive before shutdown
Remove-PhysicalDisk -PhysicalDisks $NVME -StoragePoolFriendlyName
Data -Confirm:$false
```

Contributors

Contributors to this document include:

- Alex Zarenin, Senior Solution Architect, Amazon Web Services

Further Reading

For additional information, see:

- Amazon Elastic Block Store (EBS)
- Maximize EBS volume performance with Fast Snapshot Restore
- Microsoft SQL Server on AWS
- Maximizing Microsoft SQL Server Performance with Amazon EBS
- Amazon EC2 Instance Store

Document Revisions

Date	Description
July 2020	First publication