Hyper-V VM Fleet for S2D

Dan Lovinger
Windows Server Foundation
7/18/2016 – TP5/RTM v 0.6

1 Introduction

This document describes a mechanism for running a distributed set of VMs – the 'fleet' – to provide functional or performance stress against a Storage Spaces Direct deployment. This work originated from the Windows Server 2016 TP2-aligned Ignite Conference demo from Jose Barreto and Claus Joergensen, then extended for the Intel Developer Forum 2015 demo[1] presented in mid-August 2015.

The VM Fleet is specifically adapted to performance analysis work, and allows an analyst to inject near real-time changes to the load as simply as editing and saving a script. This same mechanism can be used to create a simple demonstration environment that loops through a set of scripts.

This work is a prototype in progress, and presumes a hyper-converged deployment where the fleet VMs run on the same hardware as the Storage Spaces Direct cluster. There are certain assumptions that will require adaptation to run in generalized environments, which will be highlighted throughout the document.

Recent updates:

0.5:

- generalized sweep mechanics
- new cpu targeting sweep using StoragQoS to provide estimated IOPs per %CPU
- new helper get-linfit: provide linear fit coefficients of results in csv/tsv form

0.6:

- install-vmfleet provided to automate basic setup tasks (CSV mounts, directory structure, etc.)
- convert get-linfit.ps1 into a utility script
- add non-linearity/saturation detection to analyze-cputarget.ps1
- large fleet adaptations: clear-pause no longer scrubs pause acknowledgement files, check-pause shows the unpaused VMs.
- demo scripting works again, and now autopicks any run-demo*.ps1 script and autofills node and vm/node counts (assuming even distribution)
- flag files (pause/go/done) pushed down to a separate subdirectory of control, flag
- watch-cluster now handles downed nodes gracefully
- various smaller bugfixes

VM Fleet is now part of DISKSPD. See Section 6.

[1] See Claus Joergensen's blog for a description of the demo, here: http://blogs.technet.com/b/clausjor/archive/2015/08/18/microsoft-and-intel-showcase-storage-spaces-direct-with-nvm-express-at-idf-15.aspx

2 VM Fleet

The current scripting contains a few assumptions based on the environment it was developed in.

- that the VMs do not need external network connectivity
- that the central control point is within CSV
- location of central control point within CSV

The basic design is to have create a fleet of VMs which autologin and launch a master control script which connects back to a known location in CSV, courtesy of a loopback through an internal vswitch to their host. This script then launches the most-current load run script present, and monitors for updates and/or fleet pause requests.

SCRIPT	NOTES
LAUNCH-TEMPLATE.PS1	Per VM autologin script template: launches master.ps1, below, in a loop. ***Contains plaintext credentials when injected into the VMs.***
MASTER.PS1	Master control script for the VM. Copies in a toolset, runs load, monitors for master control and load run script updates, watches for the pause and sweep epochs.
CHECK-PAUSE.PS1	From the control console, checks how many pause acknowledgements have been received/host node. Enumerates non-paused VMs.
CLEAR-PAUSE.PS1	From the control console, clears a pause flag.
SET-PAUSE.PS1	From the control console, sets the pause flag.
INSTALL-VMFLEET.PS1	One-time script to install the vmfleet tools and create the control directory structure (see Section 0).
CREATE-VMFLEET.PS1	Creates the per-node internal VM switches and deploys the VM Fleet VMs from a pre-created VHD master image.
SET-VMFLEET.PS1	Adjusts the number of VPs and memory size/type per VM.
DESTROY-VMFLEET.PS1	Removes all vmfleet VM content.
CHECK-VMFLEET.PS1	From the control console, checks the operational state of VMs hosted throughout the cluster.
START-VMFLEET.PS1	From the control console, launches all VMs currently in **OFF** state.
STOP-VMFLEET.PS1	From the control console, shuts down all VMs currently not in **OFF** state.
RUN.PS1	A standalone load run script. This specific form is simply an example, and can be anything.
RUN-DEMO-100R.PS1 **RUN-DEMO-9010.PS1** **RUN-DEMO-7030.PS1**	These are example scripts used to set up performance demonstration environments. The demo.ps1 script, described below, causes the VM fleet to alternate between these.
RUN-SWEEPTEMPLATE.PS1	Template file for automated sweeps.
START-SWEEP.PS1	Control script for automated sweeps. See Section 6.
WATCH-CLUSTER.PS1	This is an example of text-console performance monitor tracking across a cluster. It displays the CSVFS IOP, bandwidth and latency counters, and aggregates them per-node and whole-cluster.
UPDATE-CSV.PS1	This script is used to manage the placement of tenant CSV volumes and VMs per a naming convention, described in Section 3.1 below.
DEMO.PS1	An example script to run a looped demo load with Storage Quality of Service. Run alongside *Watch-Cluster*. This assumes a set of run-demo-*.ps1 scripts (such as those included above) and a specific set of QoS policies created ahead of time: SilverVM, GoldVM and PlatinumVM. See Section 5 for example definitions.

SET-STORAGEQOS.PS1	A wrapper for Set-VMHardDiskDrive, which takes a predefined Storage QoS Policy and applies it to all VMs within the hyperconverged cluster.

2.1 Master Control

To see the master control, load runner and pause in action, connect to one of the VMs. The color splash should help make the running operation self-describing. If an issue occurs, simply ^C back to the powershell prompt (note -noexit in the launch parameters) and debug/restart the launch script, or simply shut down and reboot the VM.

3 Standing Up the VM Fleet

The VM Fleet operates on VMs installed across:

- a set of one or more CSV created per cluster node for its VMs, with virtual disks (and as a result, CSVs) named following the pattern of **<node name>[-suffix]**.
- with tools located in a VD/CSV named “collect”

These CSVs are mounted in C:\ClusterStorage per the friendly name. This convention simplifies a few tasks:

- moving CSVs within the cluster and back to their nominal owner node
- for a given host, finding its nominally owned CSV

This has proven effective in eliminating the need for additional configuration documentation, such as an XML description of the mappings. VMs are named following a similar pattern: **vm-[-group/virtual disk suffix]-<nodename>-<number>**

3.1 Create the CSV

The following fragment is an example of creating CSVs following the node naming convention, assuming that the Storage Spaces Direct pool is named per the default used by ***Enable-ClusterS2D***. Any appropriate CSV filesystem type, size or resiliency can be used.

```
Get-ClusterNode |% {
    New-Volume -StoragePoolFriendlyName S2D* -FriendlyName $_ -FileSystem CSVFS_ReFS -Size 1TB
} New-Volume -StoragePoolFriendlyName S2D* -FriendlyName collect -FileSystem CSVFS_ReFS -Size 1TB
```

3.2 Install the VM Fleet Scripts

Copy the VM Fleet scripts into a cluster node and run the ***install-vmfleet*** script. This will:

- set the well-named mountpoints in C:\ClusterStorage
- create the basic directory structure within C:\ClusterStorage\collect
  - control : the location for the scripts
  - control\tools : the location from which VMs will pick up additional tool content
  - control\result : the location to which VMs will copy their load results
- copy the scripts into control
- add control to the current user’s path
- set fleet pause (set-pause)

3.3 Create The Fleet VM Image

The VM fleet has been built focusing on the Server Core image for the guest VMs. The only requirement is that the administrator password has been set.

To construct the image, create a VM on a Server Core VHDX or install a Server Core VM using ISO media. Once installed, launch the VM and follow the prompts to specify the administrator password. This password will be specified later to the create-vmfleet script which deploys the VMs. Then tear down the VM – the resulting VHDX will be used as the base for deployment in Section 3.4.

It should be possible to use full SKUs. The most immediate change is that they will need to use shell startup items to run the launch script that is injected by the specializer. This mechanism is not used on Core since it lacks the shell.

Support and/or documentation for Nano is possible in a future update.

3.4 Create the Fleet VMs

The ***create-vmfleet*** script provisions VMs for the fleet.

By default, a dynamic VHDX will be converted to a fixed VHDX before using it for provisioning. This eliminates certain warmup effects that would occur if the VHDX remained dynamic or if the VHDX was converted to a fixed VHDX at its destination. If there is a specific measurement or deployment goal, though, it is reasonable to deploy with dynamic/unseasoned content and drive through the warmup effects that that mode of operation implies.

The script performs the following steps. It is idempotent, i.e. it can be rerun if failures occur to complete the specified deployment.

- deploys one internal vmswitch per node with the IPv4 APIPA IP 169.254.1.1; this will be the connectivity for the VMs back through the host
- copies a gold/base VHD per VM
- instantiates a VM over that VHD
- instantiates a clustered VM role for the VM
- specializes the VM/VHD
    - sets up autologin of the administrative account
    - installs VM fleet launch scripting
    - creates a sample load file for DISKSPD
    - creates an identification file naming the VM (c:\vmspec.txt)

To prepare for deployment, provide access to the VHD prepared as specific in Section 3.3. Create-vmfleet has the following switches:

- basevhd : the path to the prepared VHD
- vms : the number of vms per node per csv (group) to create
- group : specify an explicit group; else (default) it is inherited from the suffix of the CSV virtualdisk friendlyname, i.e.: <nodename>-**<suffix>**
- adminpass : password for the VM-local administrative user
- admin : (default: administrator) name of the VM-local administrative user
- connectpass : password for the user to establish the loopback connection to the host
- connectuser: name of the user to establish the loopback connection to the host
- stopafter : (***not normally needed***) used for triage, halts deployment at a specific step for each VM
- specialize : (***not normally needed***) specifies whether specialization should
    - auto : (default) be done as each VM is deployed
    - none : not be performed

    - force : always be performed, even if the VM is already deployed
- fixedvhd: (***not normally needed***) if $false, allows the base VHD to be dynamic (see **NOTE**[2])

Note that specialization requires that the VM be offline. If the VM is online, the specialization process cannot mount the VHD to inject the content. The master.ps1 script and default load file is placed at C:\run in the VM VHDX.

If an alternate network configuration is desirable, update create-vmfleet and ensure master.ps1 is modified as needed so that the VMs can establish their loopback connection to their host to CSV.

The default VM sizing follows the defaults for the New-VM cmdlet. To set a specific VM sizing, use the ***set-vmfleet*** script. Its options follow the Set-VM cmdlet:

- ProcessorCount : VP count per VM
- MemoryStartupBytes : memory reserved at VM startup
- MemoryMaximumBytes : maximum memory per VM
- DynamicMemory : (default: no, i.e. StaticMemory) whether dynamic memory is enabled

The Azure A-series VM sizes provide a baseline for VM sizing.

https://azure.microsoft.com/en-us/pricing/details/virtual-machines

4 Run the VM Fleet

At this point, the VM fleet should be ready for operation.

In addition to the pause/stop/start scripts, the basic control mechanism involves the VMs watching for an updated run*.ps1 script (note the wildcard) in the control directory. The master script checks every five seconds for pause or run script updates, so any changes should propagate in near real-time to the fleet.

5 Storage QoS

Storage Quality of Service is a new capability for Windows Server 2016. To use this with the VM Fleet, define one or more per-VM polices. Examples:

```
New-StorageQosPolicy -Name SilverVM -MaximumIops 500 -PolicyType Dedicated
New-StorageQosPolicy -Name GoldVM -MaximumIops 5000 -PolicyType Dedicated
New-StorageQosPolicy -Name PlatinumVM -MaximumIops 10000 -PolicyType Dedicated
```

Earlier versions of Windows Server 2016 Technical Previews used the policy type name `MultiInstance` instead of `Dedicated`. Both refer to a policy whose rate limit applies to each VM individually.

These names correspond to those used within the demo.ps1 demonstration script. They individually specify a range of 20x (500 – 10,000) IOP controls to put on the VMs. To then apply these policies to the VMs, use the set-storageqos.ps1 script.

[2] **IMPORTANT NOTE**: if a dynamic VHDX is used as the base and it is promoted to a fixed VHDX, the expanded region is a sparse (but reserved) hole in the ReFS allocation map for the file. In order to avoid warmup effects of committing allocation for the hole, the load file in the VM VHDX should be pre-seasoned prior to use unless these warmup effects are the specific goal of any measurements. Large sequential IO is preferred for seasoning.

6 Automated Sweeps

As of version 0.4, VM Fleet has an initial set of mechanics for automated performance sweeps. This is based on an epoch ask/response similar to how pause works.

- master tracks a "go" epoch and listens for a "done" response from run scripts
- when "done" is received, master drops a done flag file indicating the "go" epoch
- start-sweep uses this interaction to step the fleet through a sequence of run scripts, ensuring that the fleet has successfully completed each step before moving to the next
  - generated off of run-sweeptemplate.ps1
  - place into run-sweep.ps1 for each step

With version 0.5, ***start-sweep*** now accepts all parameters on the command line:

- runtemplate: the template run script which will have substitutions applied to it to produce the run file for each step (default: c:\clusterstorage\collect\control\run-sweeptemplate.ps1)
- runfile: the name to use for the run file itself (default: c:\clusterstorage\collect\control\run-sweep.ps1)
- labeltemplate: the list of sweep parameters to append together to create unique result filenames (default: b, t, o, w, p, addspec – this will generally not need to be modified)
- pc: a list of performance counters to capture in the host (ex: "\Hyper-V Hypervisor Logical Processor(*)*")
- parameters corresponding to DISKSPD:
  - b: list of buffer sizes (KiB)
  - t: list of thread counts
  - o: list of outstanding IO counts
  - w: list of write ratios
  - p: list of patterns (random: r, sequential: s, sequential interlocked: si)
  - warm: duration of pre-measurement warmup (seconds)
  - d: duration of measured interval (seconds)
  - cool: duration of post-measurement cooldown (seconds)
- addspec: inserts an arbitrary string into the result file name, to distinguish multiple sweeps which vary external parameters not under start-sweep's control. For instance: CSV placement, VCPU counts, VM numbers, and so forth.

The combination of parameters specified as lists (btowp) results in the given number of variations being run. For instance, 4, 8, and 64 KiB buffer size and 0, 10, and 30% write ratios would result in 3 x 3 = 9 variations if all other parameters were specified with single values (1 thread, etc.).

If additional controls are needed on DISKSPD, or any other tooling, simply add corresponding parameters to start-sweep, modify how the [variableset] is populated and possibly extend the default label template. Look for this block to locate the code to modify:

```
############################
############################
## Modify from here down
############################
############################
```

The default sweep template will drop DISKSPD XML results into the collect\control\result folder, which must already exist. It is currently a flagged error if any VM indicates early completion of a step, so ensure that the result folder does not have pre-existing results.

6.1 CPU Target Sweep

The CPU Target Sweep is the first higher order sweep using the sweep mechanics.

S2D, like many similar systems, will generally become CPU limited when doing small IOs. While the resulting absolute performance statements can be an exciting measure of system efficiency – 1-2 million IOPS or more! – they are not results which a realistic application (or set of VMs) could achieve since they would have little leftover CPU to drive the system.

This is where the CPU Target Sweep comes in. Using Storage QoS (see Section 5**Error! Reference source not found.**) and a simple linear extrapolation, it drives the system to a specific %CPU utilization over 100% 4KiB read and 90:10 and 70:30 r/w.

The script, ***sweep-cputarget***, takes two parameters:

- outfile: name of the result tab-seperated-value file
- cputargets: list of specific %cpu targets (ex: 20, 40, 60)

CPU is targeted +/- 5%. In practice these should indeed be linearly related through 60%. Take care in extrapolating to high average CPU utilization since saturation introduces non-linear (limiting) effects.

The ***analyze-cputarget*** script produces a report containing linear fits[3] (y = a + bx) based on the output of the sweep:

- csvfile: name of the cputarget result file
- zerointercept: whether the equations should be forced to state average CPU = 0 implies IOPS = 0 (default: false)

The zero intercept option should be used with care. A non-zero constant usually results in a better fit in the middle ranges of CPU utilization most relevant here: 20% CPU for storage, 80% CPU for VMs/Applications, and the like.

In general, 15%-80% should be relatively well handled with a linear relationship, though it is your responsibility to understand these results and make sure they are applicable to your scenario & system.

Alternate sweeps – different r/w mixes, buffer sizes – can be done by editing sweep-cputarget. This may be generalized in a future release based on feedback.

6.1.1 Examples

Graphing the results can be helpful. For instance, on a reference system we see the following using Excel's built-in trend line support (not using a zero intercept):

[3] The method used is ordinary least squares: https://en.wikipedia.org/wiki/Simple_linear_regression

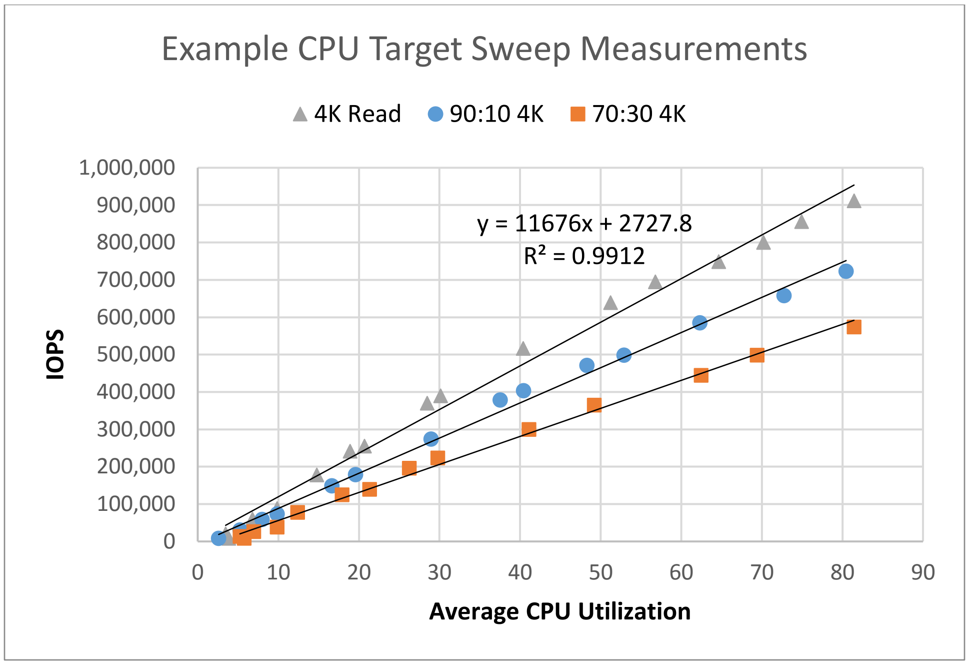


Figure 1 – Example of linear fit – only a good approximation

Note that the linear fit is only a reasonable approximation of the actual measurements. The equation and R^2 in the chart is for the 4K Reads.

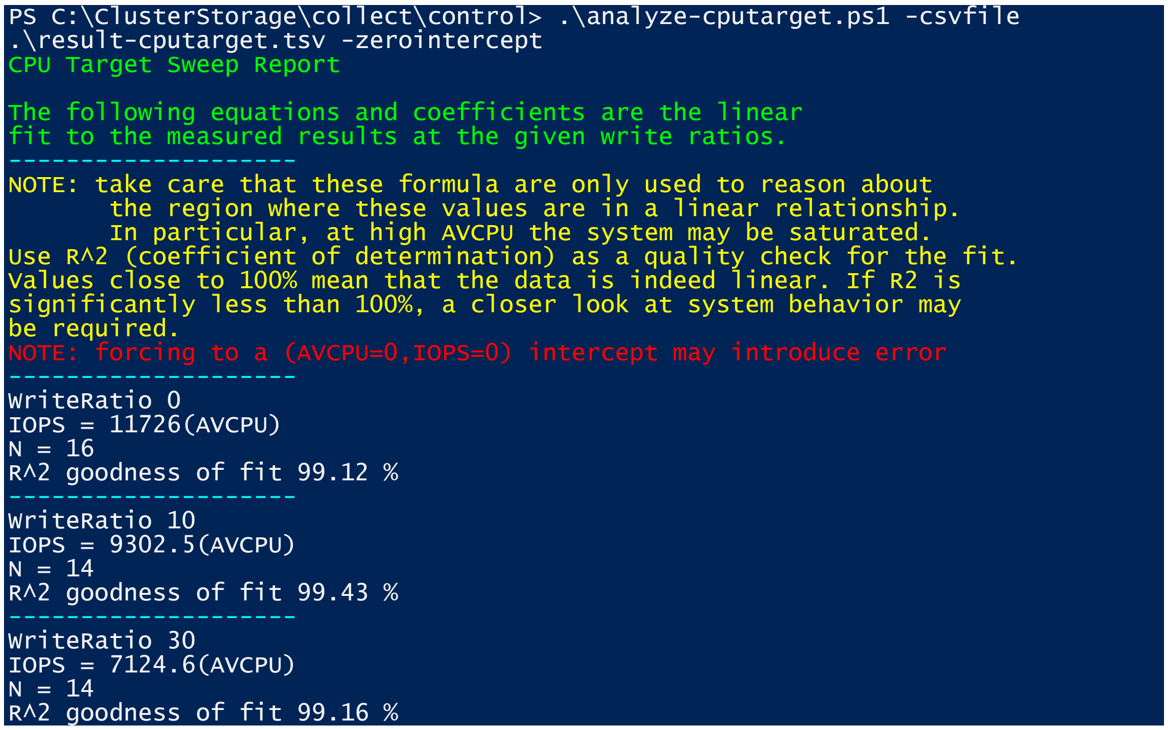
This is the equivalent analysis with analyze-cputarget. Note that the coefficients for write ratio = 0 (100% reads) and R^2 match those in the chart above. This R^2 is an example of a good fit, as the chart above should suggest. Significantly lower values of R^2 should prompt a closer look at the results, and may reveal that the system is performing in an unexpected way somewhere in the measured range.

```
PS C:\ClusterStorage\collect\control> analyze-cputarget.ps1 -csvfile .\result-
cputarget.tsv
CPU Target Sweep Report

The following equations and coefficients are the linear
fit to the measured results at the given write ratios.
--------------------
NOTE: take care that these formula are only used to reason about
       the region where these values are in a linear relationship.
       In particular, at high AVCPU the system may be saturated.
Use R^2 (coefficient of determination) as a quality check for the fit.
Values close to 100% mean that the data is indeed linear. If R2 is
significantly less than 100%, a closer look at system behavior may
be required.
NOTE: with a non-zero constant coefficient, care should be used at
low AVCPU that the result is meaningful
--------------------
WriteRatio 0
IOPS = 11676(AVCPU)+2727.8
N = 16
R^2 goodness of fit 99.12 %
--------------------
WriteRatio 10
IOPS = 9412.9(AVCPU)-5801.8
N = 14
R^2 goodness of fit 99.45 %
--------------------
WriteRatio 30
IOPS = 7500.9(AVCPU)-19003
N = 14
R^2 goodness of fit 99.56 %
```

For instance, we could estimate that at 40% average CPU utilization for 70:30 4K IO we would get 7500.9 * 40 - 19003 = about 281,000 IOPS.

With zerointercept:



```
PS C:\ClusterStorage\collect\control> .\analyze-cputarget.ps1 -csvfile
.\result-cputarget.tsv -zerointercept
CPU Target Sweep Report

The following equations and coefficients are the linear
fit to the measured results at the given write ratios.
--------------------
NOTE: take care that these formula are only used to reason about
       the region where these values are in a linear relationship.
       In particular, at high AVCPU the system may be saturated.
Use R^2 (coefficient of determination) as a quality check for the fit.
Values close to 100% mean that the data is indeed linear. If R2 is
significantly less than 100%, a closer look at system behavior may
be required.
NOTE: forcing to a (AVCPU=0,IOPS=0) intercept may introduce error
--------------------
WriteRatio 0
IOPS = 11726(AVCPU)
N = 16
R^2 goodness of fit 99.12 %
--------------------
WriteRatio 10
IOPS = 9302.5(AVCPU)
N = 14
R^2 goodness of fit 99.43 %
--------------------
WriteRatio 30
IOPS = 7124.6(AVCPU)
N = 14
R^2 goodness of fit 99.16 %
```

In this case we would estimate that at the same 40% average CPU utilization for 70:30 4K we would get 7124.6 * 35 = about 285,000 IOPS. While this seems like the more natural relationship, it is likely to be somewhat farther off.

7 Further Resources

For information on Storage Spaces Direct (S2D), including deployment instructions: http://aka.ms/s2d

To download a DISKSPD binary and its documentation: http://aka.ms/diskspd

VM Fleet is now part of DISKSPD. DISKSPD is OSS, hosted on GitHub: https://github.com/microsoft/diskspd

The binary distribution may lag the OSS repo from time to time. The most up to date versions of DISKSPD and the VM Fleet will be found at GitHub.