

Effects of “Virtual Services” in App-V 5 SP2 with HotFix4 Deployment Performance

TMurgent App-V 5 Performance Research Series



Photo: Courtesy Watcharakun

June, 2014

Table of Contents

1	Introduction.....	4
2	Background on Windows and Services.....	5
2.1	Updater Services	5
2.2	Services to use elevated permissions.....	5
2.3	Services to support multi-user on the OS	5
2.4	Services with Protection	6
2.5	Services to support multiple users on different machines.....	6
2.6	Services to support multiple applications.....	6
2.7	App-V 5 and the Virtual Services Subsystem	6
2.8	Sequencer detection of Services and support	7
2.9	App-V 5 and Service Limits	8
2.10	Detecting Services inside a Package	8
2.11	Services and Scripts.....	8
3	Summary of Where Impacts of Virtual Services Are Felt.....	10
3.1	Standard Scenario Testing results.....	10
3.2	Results Interpretation	13
3.3	Improvement Options.....	13
4	Testing Strategy Used	14
4.1	About the Testing Platform.....	14
4.2	About Test Packages and “Streaming Configuration”	14
4.3	About the Testing Methods	14
4.3.1	Test Package.....	15
4.3.2	Test Pass.....	15
4.3.3	Test Cycle	16
4.4	About the Test Results Accuracy.....	16
5	Test Packages Utilized.....	17
5.1	Warmup Package	17
5.2	Lots_OfNothing (Baseline).....	17
5.3	Lots_OfServices_Disabled.....	18
5.4	Lots_OfServices_Manual	18
5.5	Lots_OfServices_AutoStart	18
5.6	Anatomy of a Package with a Service.....	19
6	Detail Test Results.....	21

6.1	SCS Mode Without Mounting.....	22
6.2	SCS Mode Testing with Mounting	24
6.3	SCS Mode Disabled, No Mounting	25
6.4	SCS Mode Disabled, Mounting Used.....	26
7	About This Research Paper Series.....	27

1 Introduction

The purpose of this research paper is to document the effects that virtual application services have in Microsoft App-V Virtual Application Packages.

The effort is squarely aimed at answering questions on how the addition of virtual services affect performance, however some other interesting tidbits also get surfaced.

This work is part of a series of papers to characterize the impact that different application elements have on the performance of virtual applications under App-V 5.

Most readers of this research will find themselves satisfied with reading the second and third section of this paper. The remaining sections detail the testing process, packages used, and provide further test details and additional findings.

2 Background on Windows and Services

Windows Services are a critical portion of the Microsoft Windows Operating System. Many important portions of the OS itself are carried out by services, but sometimes applications add Windows Services to the OS as well.

Services are, in general, background processes that are not tied to a particular user logon. Except for some very old legacy services, they do not involve interaction with the user directly. The services will normally run under credentials other than user credentials, allowing the software to perform tasks that would not be permitted by standard users. Applications will sometimes include one or more services.

When sequencing an application that includes a service we can choose to let the service run as a virtual service inside the package. We can also choose to disable, set to manual start mode, or remove the service completely from the package. Possibly the service is not needed, or possibly we will choose to deploy the service natively.

2.1 Updater Services

The most common form of application installed services are those designed to check for application software updates, and to implement the updates. Because the service has special permissions, it can download the updates and perform the installation, even when the logged in user does not have installation permissions. Typically, we work hard to defeat these auto-updater services when we create App-V packages, preferring to have IT update the package once and deploy to all users in parallel instead of users randomly receiving updates to an untested version.

2.2 Services to use elevated permissions

In other cases, these application services exist to perform other operations within the application, perhaps requiring elevated permissions, and these we need to support in the App-V Package when possible.

2.3 Services to support multi-user on the OS

In a few cases, the service exists to allow multi-user functionality on a shared operating system. Typically, App-V is already providing the multi-user functionality, so we don't really need the service to do that for us. While it might be OK to disable or remove the service, most of the time the service remains required for the application to work. And usually this can be supported inside the package. But there can also be exceptions.

2.4 Services with Protection

A special class of services have also evolved that are designed for protection. These services have been designed to ensure that other software cannot impact the service software itself, and the way this has been done proves to be incompatible with App-V. Both the FlexNet licensing service (used in many applications) and Google Chrome Enterprise use DACLs on the service itself that prevent App-V from controlling the service. We usually can work around the restriction by changing those permissions after software installation. Another scenario is sometimes referred to as “sandbox mode” or “protected mode”, although this scenario might affect both the application and services. Currently, we resort to disabling these modes in order to virtualize the application.

2.5 Services to support multiple users on different machines

Some applications are designed to support multiple users on different machines, but all sharing some common data other than through an external server or file share. In these apps, typically the first user generates the common data store, possibly file based or a database, and get a windows service to allow the other users access.

In a native installation, all that is needed is for the user’s PC with the datastore to remain powered on for other users to have access. Under App-V, the service would only be running while the user was logged on and running the application so including such a service in the package is typically a bad idea. It might be deployed natively.

2.6 Services to support multiple applications

The FlexNet licensing service is included as part of license enforcement by a number of application vendors. Typically the user stands up an external centralized license server that contains the actual license data. Each application installed that uses FlexNet will install a local license enforcement service (only if not already installed on the OS) that will contact the central license server on behalf of the application. When an additional application is installed it will just use the existing service on the OS.

Once we work around the DACL issue with this service, virtualizing the service under App-V is OK as long as there is only one virtual application containing the service running on an OS at one time. But if two virtual applications have the service and are run at the same time, the second one will fail. This can be solved by installing the service natively on the OS, or by placing the service by itself as a “middleware” package to be referenced by each application in a Connection Group.

2.7 App-V 5 and the Virtual Services Subsystem

One of the specialized subsystems in App-V 5 is the virtual services subsystem. The sequencer is designed to detect the presence of services inside a package and identifies these as Service

Extension Points. Unlike many of the App-V 5 extension points, services considered “internal” extensions. Unless the service exposes an external API (such as listening on a socket), only the virtualized application will be able to access it.

Virtual services are seen as windows processes, but are not known to the external windows services manager. Unlike the native windows service that “auto starts”, the virtualized service is not running when applications in the virtual environment are shut down.

When a virtual environment for a package is not already started, the App-V Client services subsystem will hold off starting an application that contains an autostart service until the service is started. If multiple virtual services are present in the package, they will be started serially.

When all applications in the virtual package have ended, the client will then shut down all virtual services in the package. The service shutdowns appear to happen in parallel.

2.8 Sequencer detection of Services and support

A service consists of one or more files, plus an HKLM registration. Typically application services are an exe file (plus possibly other files) that run in a separate process. Services may also be designed as a dll to be loaded in a separate thread by an existing SVCHost process running in the same logon account (these will get a separate process anyway when virtualized in App-V).

Not all services are supported for virtualization. A service definition contains a startup-type field that defines how the natively installed service gets started. App-V alters how these get started, and cannot support some types:

Type	Native	Virtualized
Boot	Starts before Winlogon	Not supported
Disabled	Cannot Start	Cannot Start
Manual	When an app requests	When a virtual app requests
AutoStart	When first user logs in	When VE Starts
AutoStart-Delayed	When idle after first user logs in	When VE Starts

Microsoft states that services that are Boot-time services are not supported. Additionally, services marked for AutoStart-Delayed can only work as if marked AutoStart.

Native Services can also run under different logon accounts. App-V Services may only run under the Local System Account.

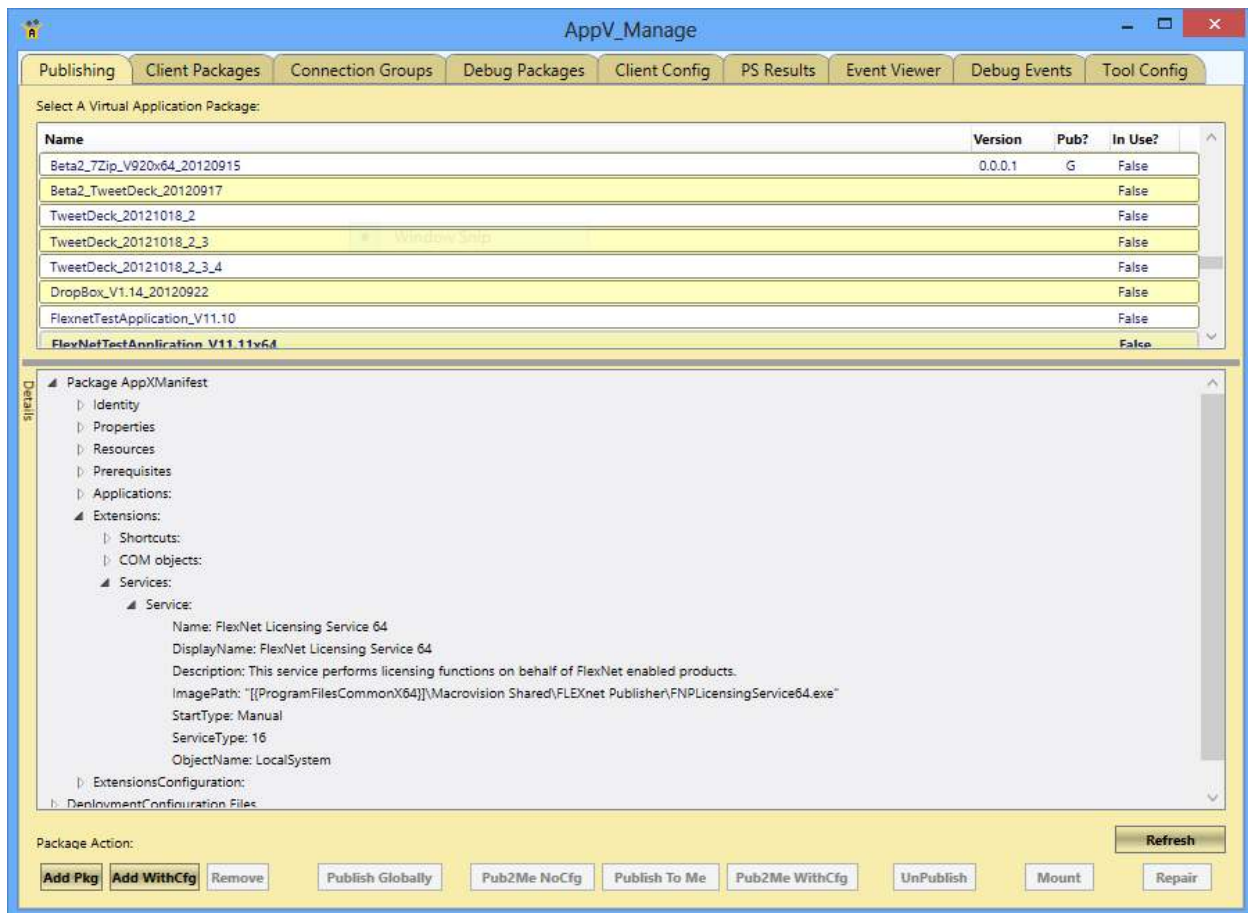
NOTE: As of 5.0 SP2, the sequencer will silently ignore services that are installed and registered to run under a logon account other than Local System. To virtualize such services, you must modify the logon account to Local System while in installation monitoring mode.

2.9 App-V 5 and Service Limits

It is not known if there is a limit to the number of virtual services inside a package. I have tested 20 virtual services inside a package successfully, and I doubt anyone needs more than that.

2.10 Detecting Services inside a Package

How do you detect services in the package? If properly captured, you can look at the services tab of the Sequence Editor. You can also view these by looking in the DeploymentConfig or internal AppXManifest file for the App-V package. One way to view the internal AppXManifest file is to use the AppV_Manage¹ tool:



When a service is not captured by the sequencer, such as due to an incorrect logon account or startup type, no warning is (currently) issued. So you have to manually detect these.

2.11 Services and Scripts

In addition to starting AutoStart virtual services at the start of a virtual environment, the App-V client might also process a Start of Virtual environment script. An analysis of this behavior shows

¹ AppV_Manage is a free tool for use on Test Client systems. You may download it from www.tmurgent.com.

that the Client subsystem begins starting autostart services before processing scripts. With a typical number of virtual services, these services will start prior to the script. However, with a larger number the script can start prior to all of the services starting.

When all of the other processes of the virtual environment end, the virtual services are terminated in parallel prior to the end of virtual environment script. Even with 20 virtual services running, because the services are terminated in parallel the script runs after the last service completes.


3 Summary of Where Impacts of Virtual Services Are Felt

This section highlights the most important results. Additional details appear in subsequent sections, however many readers will stop reading after this section.

3.1 Standard Scenario Testing results

Virtual Services have the second most significant impact on deployment performance measured in this series. While removal of unnecessary services outright will improve package performance the most, changing the start mode from AutoStart to either Manual or Disabled will at least help the runtime aspects.

Virtual Services have an impact on the Add and Publish steps of deployment. The service will include some registry settings and a folder or two plus files. The small increases in size of the Central Directory, Virtual Registry file, AppXManifest file, and BlockMap file will cause packages with virtual services of any type to slow down these two steps. The difference, for an average service is barely detectable for the Add step and not detectable in the Publishing step.



VIRTUAL SERVICES IN A PACKAGE SET FOR DEMAND START OR DISABLED HAVE A SMALL EFFECT ON THE PACKAGE.

Virtual Services set for AutoStart do have a significant affect the First and Subsequent Run attempts, while those set for Demand Start or Disabled are negligible. The amount of the impact depends upon the design virtual service itself, as it is left to the service to signal that it is ready. A simple service will signal this quickly, while a more complicated service might take a while longer to load and initialize.

There is slightly more overhead in the first run than subsequent runs for these autostart services, but the bulk of the overhead affects all runs of the virtual environment.

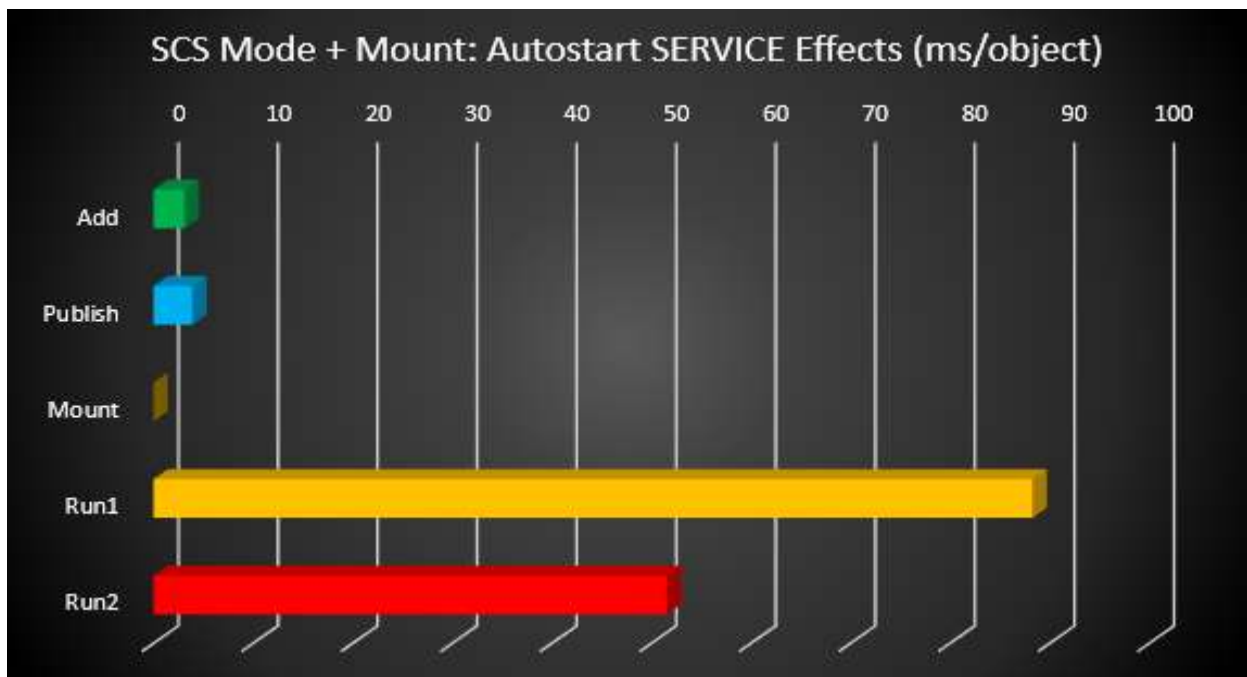
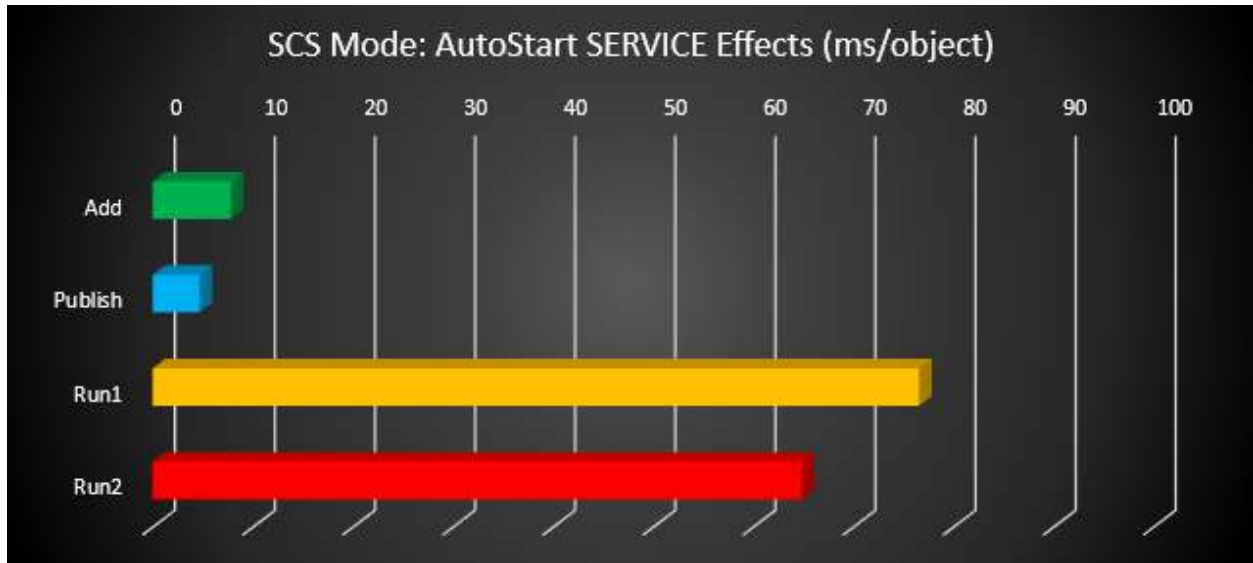


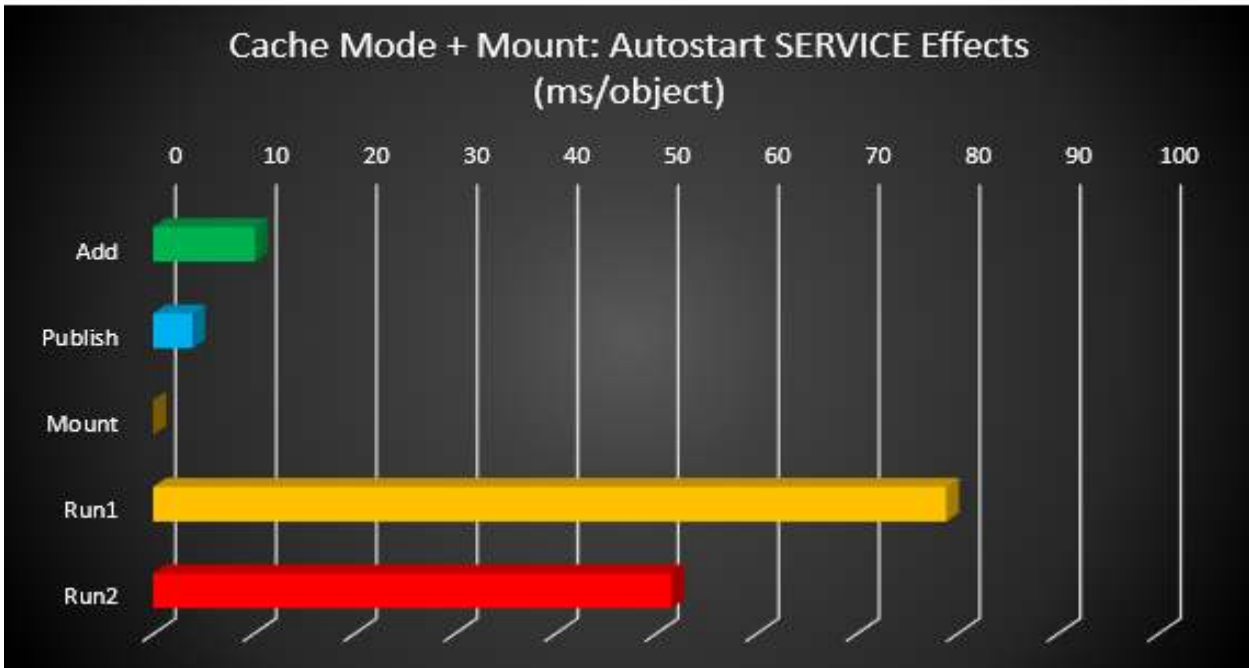
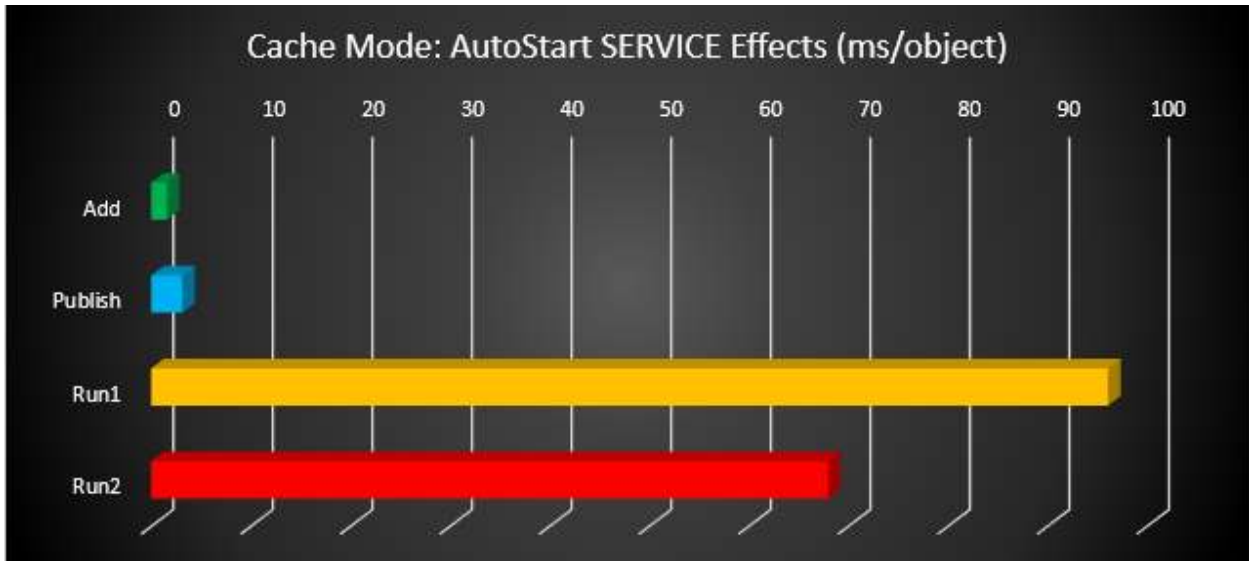
VIRTUAL SERVICES ARE STARTED SERIALLY, NOT IN PARALLEL.

From the tests, we can see that virtual services are started serially, meaning the delay in launching the application is the sum of this delay for all services. Virtual services appear to be shut down in parallel.

The results presented that follow are for a very minimal service executable that will behave equally or better than most services. The results shows are the increase in time for launching 20

simple services with AutoStart over that of an empty package, divided by the 20 services to achieve an impact per service. Run times include both time to start up and shutdown any running virtual services in the tests.





With only 20 Services included in the testing for this paper, the results are more subject to variation in background activity than other papers in this series. Individual results should be considered to be +/- 2ms/service. For example, a 1ms/service difference in a result could be the effect of a single windows clock tick.

3.2 Results Interpretation

Virtual Services have almost no impact to the Add and Publish phases of deployment. Any difference is due to the added files and registry and subsequently larger xml files, and is the same no matter how the service is configured.

The results for the first run “Caching with Mount” stand out in the tests as unusually long. This is especially true when viewed against the “Caching No Mount”. I did not determine the cause of the added delay for the first run when files are already present in the cache.

In “SCS with Mount” case, caching the package locally performed a little better than without, but not that much. The second run, however, exhibited the best performance of all tests.

3.3 Improvement Options

The following options for working with packages that include services:

1. Removing unnecessary services will always provide the best performance. However, the difference between leaving a service intact and disabling it versus removal is extremely small. It is probably best to just disable the service, just in case there is a scenario where the application looks for the service and fails if not present.
2. Either disabling or removing a virtual service that would otherwise be set for AutoStart saves a minimum of 40ms per service to the startup/shutdown time. In certain test scenarios, I saw up to nearly 80ms/service difference for a small and benign service.
3. Services that perform significant functions will have even greater impacts.
4. Pre-caching appears to improve launch performance, except when it doesn't.
 - Pre-caching generally improves launch performance, however with virtual services there can be exceptions that affect first run performance.
5. Disabling the virtual services may also be considered. This may be done as a post-sequencing operation by editing the DeploymentConfiguration.XML file and only affects the one package. This change carries the same or slightly higher risks than removing the services.

4 Testing Strategy Used

This section provides details about how the testing was performed.

4.1 About the Testing Platform

The testing results depicted in this paper are based on:

App-V 5.0 SP2 with Hotfix 4 running on a Windows 7 SP1 x86 virtual machine.

The testing was performed in an isolated environment using a Microsoft 2012 R2 server with Hyper-V. The server has 24 processors and 64GB of RAM. To minimize external impacts, this server utilizes local storage and contains a VM with the domain controller. App-V Package sources were located on a share on this host.

The Test VM used had 2GB of RAM and was given 2 virtual CPUs. The App-V Client is configured for Shared Content Store mode (which disables background streaming).

4.2 About Test Packages and “Streaming Configuration”

All Test packages used are specially constructed software packages that I developed. These packages are generally stripped down to a bare minimum, except for an overabundance of the one particular things we want to measure when using this package. In many cases, this means custom software that I developed for the purpose of the test.

Unless specifically noted, each package was sequenced and configured for streaming by *not* launching anything during the streaming training configuration phase of the sequencer. This means that, barring mounting operations, almost everything in the package will fault-stream (stream on demand).

4.3 About the Testing Methods

All tests are automated using significant sleep periods before each portion of the testing to allow all systems to settle down, and warm-up of the external components (hypervisor/fileshare) and within the OS (App-V Client and drivers) are performed. The test process consists of:

- A **Test Cycle** that consists of a series of Test Passes.
- Each **Test Pass** consists of a number of Test Packages.
- Each **Tested Package** is tested using a series of actions and measurements.

A **Tested Package**, consists of a series of actions, always preceded by a significant sleep period to allow system background processes to settle down.

A **Test Pass** always starts from a freshly booted snapshot and with a dummy **Test Package** to warm up the App-V Client and Driver sub-systems. The results of this dummy package are not used.

A *Test Cycle* always starts with a *Test Pass* to warm up the external components of the Hypervisor and Windows File Share. Because the packages are relatively small compared to the amount of memory available, the packages are likely retained in memory in the Windows Standby Lists after the initial *Test Cycle*. These are described as follows, from the bottom up.

4.3.1 Test Package

For a given ***Test Package***, the series of actions includes:

- Waiting
- Add-AppVClientPackage
- Waiting
- Publish-AppVClientPackage
- Waiting
- [Optionally Mount-AppVClientPackage²]
- Waiting
- First run (launch “cmd.exe³ /c time /t” inside the virtual environment).
- Waiting
- Second run (launch “cmd.exe⁴ /c time /t” inside the virtual environment).

The time required for each of the actions to complete is recorded.

4.3.2 Test Pass

A ***Test Pass*** consists of testing multiple *Test Packages* as follows:

- Reverting the test VM to a snapshot.
- Waiting for the Hypervisor to settle.
- Booting the VM and logging in.
- Waiting.
- A series of actions and measurements on a warm-up package. These results are never used, it is only performed to warm up the client (client service, drivers, and WMI) and to ensure that each subsequent package fairly tested under similar conditions.
- Waiting.
- A series of actions and measurements on the first package.

² With SCS enabled, mounting the package does result in the actual file content being stored in the App-V file cache. I test in SCS mode both with and without mounting to better delineate the cause of performance slowdowns on a package.

³ This is used rather than a program in the package to produce a comparable time that varies based on special actions that the client must perform during virtual environment startup and shutdown due to the package content.

⁴ The client is also known to perform special actions the first time a virtual environment is used, so the second run is used for comparison to the first run.

- Waiting.
- A series of actions and measurements on the second package.
- Etc...
- Recording results

4.3.3 Test Cycle

Finally, A **Test Cycle** consists of several consecutive test runs of the same *Test Pass*. The first pass is used to “warm up” external systems and achieve a relatively consistent amount of caching by the server. The results of this pass are not used, but the results of the remaining passes are averaged to produce results. A Test Cycle typically requires a full day to complete.

4.4 About the Test Results Accuracy

As careful as I attempt to be to eliminate variability in the results, there is a fair amount of variability in results between two passes.

Due to the nature of the background interruptions affecting the results, the impact on result accuracy is felt much more on measurements that are shorter in duration than those that are longer. With measurements that are sub-second, this can produce results that typically vary by as much as +/-10% from the average. It might be possible to wash out this effect by making a package with an extreme number of virtual services and increase the number of Test Cycles by an order of magnitude, but there does not seem to be a lot of value in doing so as the major impact shown in the testing is so much greater than these small differences.

Instead, I use an approach to test with a sufficient number of test cycles and select the minimum value seen on any of the tests. The more repetitions that are made, the better this minimum value represents the time it takes for App-V to complete the task without the effects of any extraneous background interference.

5 Test Packages Utilized

This section details the packages used in testing.

All packages used in this test with services contain a set of 20 windows services. These services are built as simple as possible, containing only enough code to support startup and shutdown interfaces. A shortcut to the cmd prompt is added to each package. The difference between the three packages is in the configuration of the service.

Each package was independently sequenced and will contain slightly different flotsam from background activity that occurs in every package. Those differences are quite small.

5.1 Warmup Package

This package is primarily used as the first package in a Test Pass, to warm up the OS and App-V Client components and dependencies⁵.

5.2 Lots_OfNothing (Baseline)

This is a minimal App-V Package.

In developing this package, I discovered that there is an issue with the App-V Client in that there appears to be some sort of undocumented minimal package requirements. If you create a package with no registry entries, no files, and no integrations, the Add-AppVClientPackage cmdlet will error out with error 700002.

Therefore this package consists of one HKLM registry key, one HKCU registry key, one text file in the PVAD folder, and one shortcut (to the text file).

⁵ When conducting tests that use mounting, I found it necessary to warm up the system without mounting this package. It appears that mounting this package causes an additional any subsequent Add-AppVClientPackage commands to take an extra around an extra second to complete. This issue only seems to exist with this package, and mounting other packages does not affect subsequent Add cmdlets. The cause of this is unknown.

5.3 Lots_OfServices_Disabled

This is a package containing 20 Windows Services, all installed and set to disabled. Each service executable is a single exe copied to the PVAD folder and installed. They perform no work and are the minimal amount of code to start up. The package contains a shortcut for the CMD prompt for debugging.

Package Statistics⁶:

Size of .AppV File	1,227,436
Size of Central Directory	3,210
Size of BlockMap (Compressed)	2,521
Size of AppxManifest (Compressed)	1,111
Size of Registry.Dat (Compressed)	52,630
Number of Entries + EmptyDirectories	31+3
Number of Services Detected	20

5.4 Lots_OfServices_Manual

This is a package containing 20 Windows Services, all installed and set to disabled. Each service executable is a single exe copied to the PVAD folder and installed. They perform no work and are the minimal amount of code to start up. The package contains a shortcut for the CMD prompt for debugging.

Package Statistics:

Size of .AppV File	1,229,332
Size of Central Directory	3,458
Size of BlockMap (Compressed)	2,683
Size of AppxManifest (Compressed)	1,117
Size of Registry.Dat (Compressed)	53,646
Number of Entries + EmptyDirectories	33+3
Number of Services Detected	20

5.5 Lots_OfServices_AutoStart

This is a package containing 20 Windows Services, all installed and set to disabled. Each service executable is a single exe copied to the PVAD folder and installed. They perform no work and are

⁶ Package Statistics are provided by a tool called "LookAtAppVFile" developed by the author. "Number of Services Detected" indicates the number recorded by the sequencer as services in the XML files; in some cases services may exist but become undetected by the sequencer, rendering them useless.

the minimal amount of code to start up. The package contains a shortcut for the CMD prompt for debugging.

Package Statistics:

Size of .AppV File	1,227,946
Size of Central Directory	3,230
Size of BlockMap (Compressed)	2,543
Size of AppxManifest (Compressed)	1,114
Size of Registry.Dat (Compressed)	53,381
Number of Entries + EmptyDirectories	31+3
Number of Services Detected	20

5.6 Anatomy of a Package with a Service

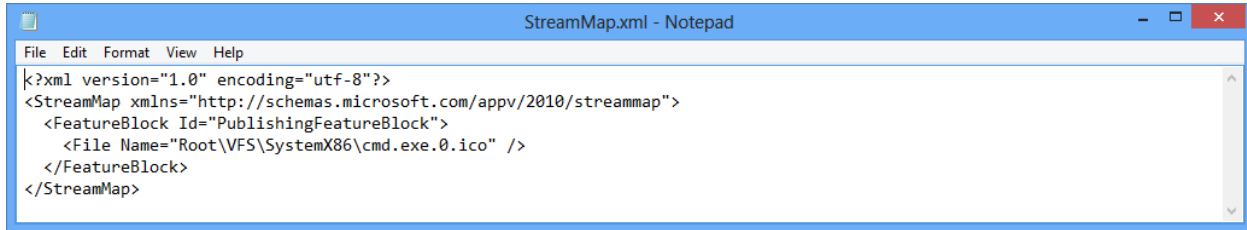
Services that are detected are listed as extensions in the internal AppXManifest and external dynamic configuration XML files. The image that follows is from the AutoStart package AppXManifest file:

```

<Applications xmlns="http://schemas.microsoft.com/appv/2010/manifest">
  <Application Id="{SystemX86}\cmd.exe" Origin="Application" TargetInPackage="false">
    <Target>{SystemX86}\cmd.exe</Target>
    <VisualElements>
      <Name>CMD for Lots_OfServices_AutoStart_PVAD</Name>
      <Version>6.1.7601.17514</Version>
    </VisualElements>
  </Application>
</Applications>
<appv:Extensions>
  <appv:Extension Category="AppV.Service">
    <appv:Service Name="LotsOfService_AutoStart_1" DisplayName="LotsOfService AUTOSTART #1"
    ImagePath="{AppVPackageRoot}\LotsOfService_AutoStart_1.exe" StartType="Automatic" ServiceType="16"
    ObjectName="LocalSystem" />
  </appv:Extension>
  <appv:Extension Category="AppV.Service">
    <appv:Service Name="LotsOfService_AutoStart_2" DisplayName="LotsOfService AUTOSTART #2"
    ImagePath="{AppVPackageRoot}\LotsOfService_AutoStart_2.exe" StartType="Automatic" ServiceType="16"
    ObjectName="LocalSystem" />
  </appv:Extension>
  <appv:Extension Category="AppV.Service">
    <appv:Service Name="LotsOfService_AutoStart_3" DisplayName="LotsOfService AUTOSTART #3"
    ImagePath="{AppVPackageRoot}\LotsOfService_AutoStart_3.exe" StartType="Automatic" ServiceType="16"
    ObjectName="LocalSystem" />
  </appv:Extension>
  <appv:Extension Category="AppV.Service">
    <appv:Service Name="LotsOfService_AutoStart_4" DisplayName="LotsOfService AUTOSTART #4"
    ImagePath="{AppVPackageRoot}\LotsOfService_AutoStart_4.exe" StartType="Automatic" ServiceType="16"
    ObjectName="LocalSystem" />
  </appv:Extension>
  <appv:Extension Category="AppV.Service">
    <appv:Service Name="LotsOfService_AutoStart_5" DisplayName="LotsOfService AUTOSTART #5"
    ImagePath="{AppVPackageRoot}\LotsOfService_AutoStart_5.exe" StartType="Automatic" ServiceType="16"
    ObjectName="LocalSystem" />
  </appv:Extension>
  <appv:Extension Category="AppV.Service">
    <appv:Service Name="LotsOfService_AutoStart_6" DisplayName="LotsOfService AUTOSTART #6"
  </appv:Extension>
</appv:Extensions>

```

Services are not automatically added to the publishing stream block, even when set to autostart. The image that follows is the StreamMap for the package containing the Autostart services:

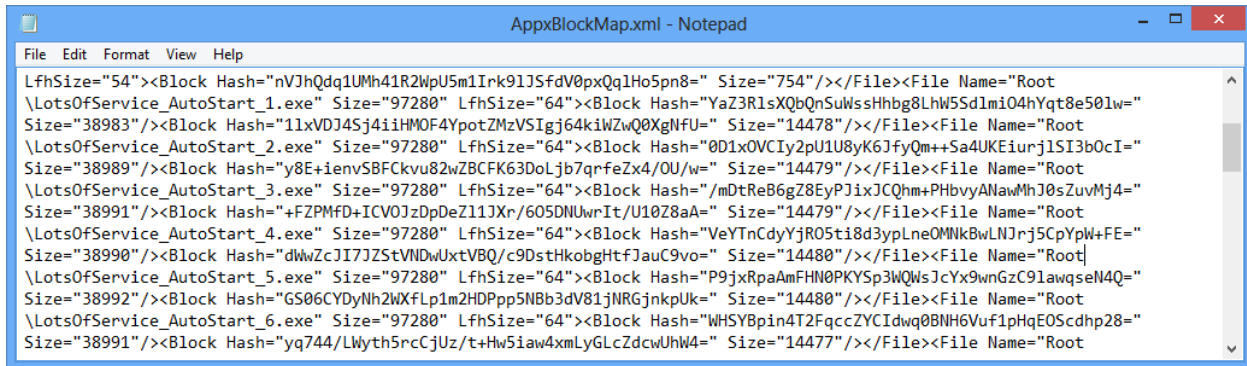


```

StreamMap.xml - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="utf-8"?>
<StreamMap xmlns="http://schemas.microsoft.com/appv/2010/streammap">
  <FeatureBlock Id="PublishingFeatureBlock">
    <File Name="Root\VFS\SystemX86\cmd.exe.0.ico" />
  </FeatureBlock>
</StreamMap>

```

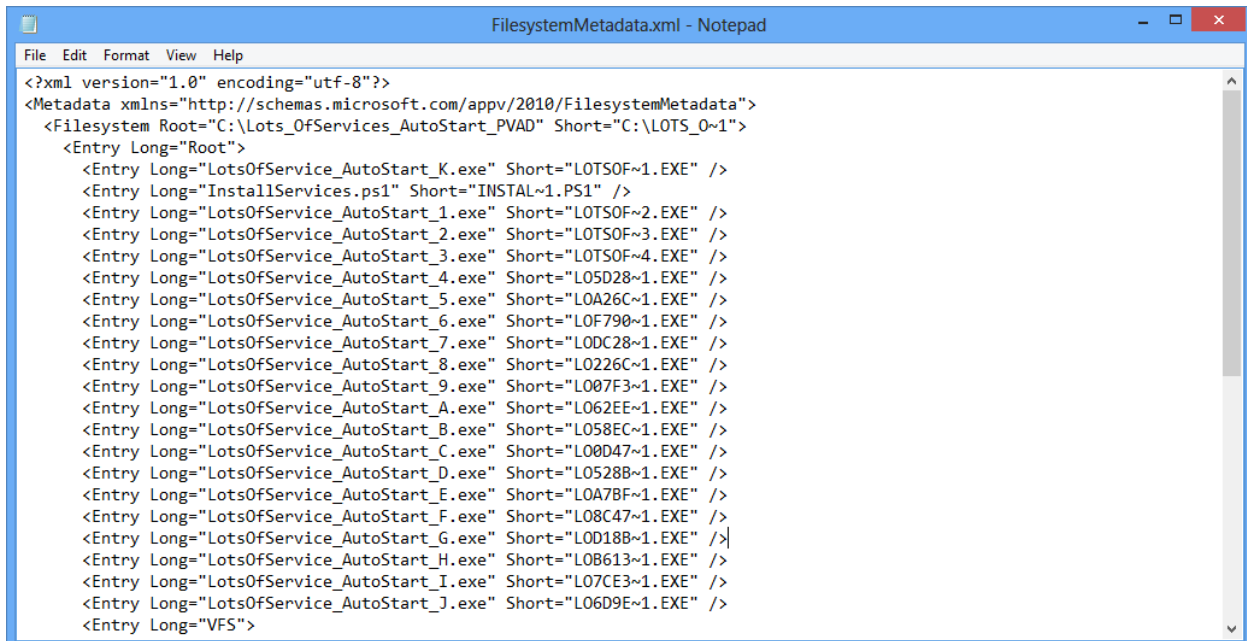
Because the services are files, they make a small increase in the size of the AppxBlockMap and the FilesystemMetadata xml files that must be read by the client. Except for complex services, this increase is insignificant. The following images are again from the Autostart package:



```

AppxBlockMap.xml - Notepad
File Edit Format View Help
LfhSize="54"><Block Hash="nVjhQdq1UMh41R2WpU5m1Irk91J5fdV0pxQq1Ho5pn8=" Size="754"/></File><File Name="Root
\LotsOfService_AutoStart_1.exe" Size="97280" LfhSize="64"><Block Hash="YaZ3R1sXQbQnSuWssHhbg8LhW5Sd1mi04hYqt8e501w="
Size="38983"/><Block Hash="11xVDJ45j4iiHMOF4YpotZMzVSIgJ64kiWzWQ0XgNFU=" Size="14478"/></File><File Name="Root
\LotsOfService_AutoStart_2.exe" Size="97280" LfhSize="64"><Block Hash="0D1x0VCIy2pU1U8yK6JfyQm++Sa4UKEiurj1SI3b0cI="
Size="38989"/><Block Hash="y8E+ienvSBFCkvu82wZBCFK63Doljb7qrfeZx4/OU/w=" Size="14479"/></File><File Name="Root
\LotsOfService_AutoStart_3.exe" Size="97280" LfhSize="64"><Block Hash="/mDtReB6gZ8EyPJixJCQhm+PHbvyanawMhJ0sZuvMj4="
Size="38991"/><Block Hash="+FZPMFD+ICV0JzDpDeZ11JXr/605DNUwrIt/U10Z8aA=" Size="14479"/></File><File Name="Root
\LotsOfService_AutoStart_4.exe" Size="97280" LfhSize="64"><Block Hash="VeYtnCdyYjR05ti8d3ypLneOMNkBwLJnrj5CpYpW+FE="
Size="38990"/><Block Hash="dWwZcJI7JZStVNDwUxtVBQ/c9DstHkobgHtfJauC9vo=" Size="14480"/></File><File Name="Root
\LotsOfService_AutoStart_5.exe" Size="97280" LfhSize="64"><Block Hash="P9jxRpaAmFHN0PKYSp3WQMsJcYx9wnGzC91awqseN4Q="
Size="38992"/><Block Hash="GS06CYDyNh2WxFLp1m2HDPpp5NBb3dV81jNRGjnkpUk=" Size="14480"/></File><File Name="Root
\LotsOfService_AutoStart_6.exe" Size="97280" LfhSize="64"><Block Hash="WHSYBpin4T2FqccZYCIDwq0BNH6Vuf1pHqEOScdhp28="
Size="38991"/><Block Hash="yq744/LWyth5ncCjUz/t+Hw5iaw4xMlyGLcZdcwUhw4=" Size="14477"/></File><File Name="Root

```



```

FilesystemMetadata.xml - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="utf-8"?>
<Metadata xmlns="http://schemas.microsoft.com/appv/2010/FilesystemMetadata">
  <Filesystem Root="C:\Lots_OfServices_AutoStart_PVAD" Short="C:\LOTS_0~1">
    <Entry Long="Root">
      <Entry Long="LotsOfService_AutoStart_K.exe" Short="LOTSOF~1.EXE" />
      <Entry Long="InstallServices.ps1" Short="INSTAL~1.PS1" />
      <Entry Long="LotsOfService_AutoStart_1.exe" Short="LOTSOF~2.EXE" />
      <Entry Long="LotsOfService_AutoStart_2.exe" Short="LOTSOF~3.EXE" />
      <Entry Long="LotsOfService_AutoStart_3.exe" Short="LOTSOF~4.EXE" />
      <Entry Long="LotsOfService_AutoStart_4.exe" Short="LO5D28~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_5.exe" Short="LOA26C~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_6.exe" Short="LOF790~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_7.exe" Short="LODC28~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_8.exe" Short="LO226C~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_9.exe" Short="LO07F3~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_A.exe" Short="LO62EE~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_B.exe" Short="LO58EC~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_C.exe" Short="LO0D47~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_D.exe" Short="LO528B~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_E.exe" Short="LOA7BF~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_F.exe" Short="LO8C47~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_G.exe" Short="LOD18B~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_H.exe" Short="LOB613~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_I.exe" Short="LO7CE3~1.EXE" />
      <Entry Long="LotsOfService_AutoStart_J.exe" Short="LO6D9E~1.EXE" />
      <Entry Long="VFS">

```

6 Detail Test Results

This section provides additional details of the testing results not reported in the summary.

Results reported are based on an ideal test environment. Performance impacts identified in this paper will be very different in production environments. Specific numbers are *only* useful in comparison to numbers from other research papers in this series!

The simplified results provided earlier in the summary are based on substantially detailed testing, some of which the results are presented here. These tests are not necessarily scenario based, but are designed to illuminate the actions of different parts of the App-V client. These details allow for a more complete understanding of where deployment time impacts come from.

6.1 SCS Mode Without Mounting

In this test, Shared Content Store Mode is enabled as would typically be used. These results show the performance that you can expect when SCS Mode is enabled.



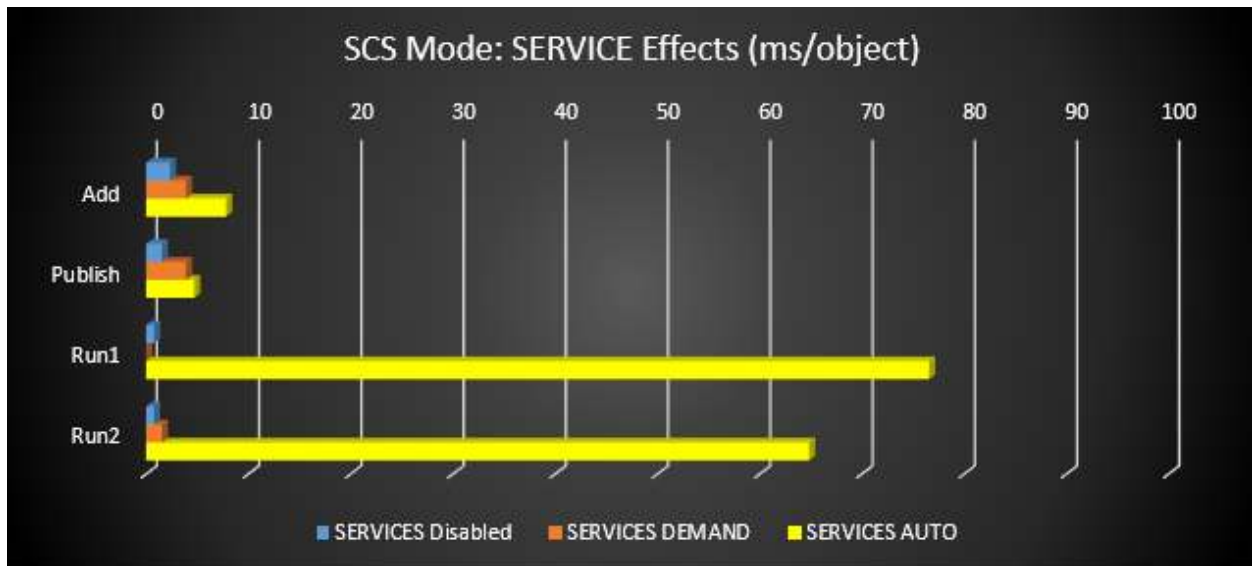
EACH SMALL VIRTUAL SERVICE ADDS 4MS TO THE ADD STEP AND 3MS TO THE PUBLISH STEP, NO MATTER HOW IT IS CONFIGURED.

These results may also be interpreted to show the performance that you can expect without SCS mode enabled when background streaming is not enabled and virtual service files are not included in the streaming configuration.



THE STARTUP TYPE OF A VIRTUAL SERVICE DOES NOT IMPACT PERFORMANCE OF THE ADD AND PUBLISH STEPS.

The chart which follows shows the results calculated from tests on the various services packages in these tests. The times shown are calculated as the time for the given package minus the time for the BasePkg divided by the number of virtual services, producing a calculated impact in ms/service for each operation.



In the Add and Publish phases, these results show that there is a fairly small amount of overhead due to the presence of virtual services, in the range of 4 to 10 ms/service. Presumably a larger service object would increase these numbers.

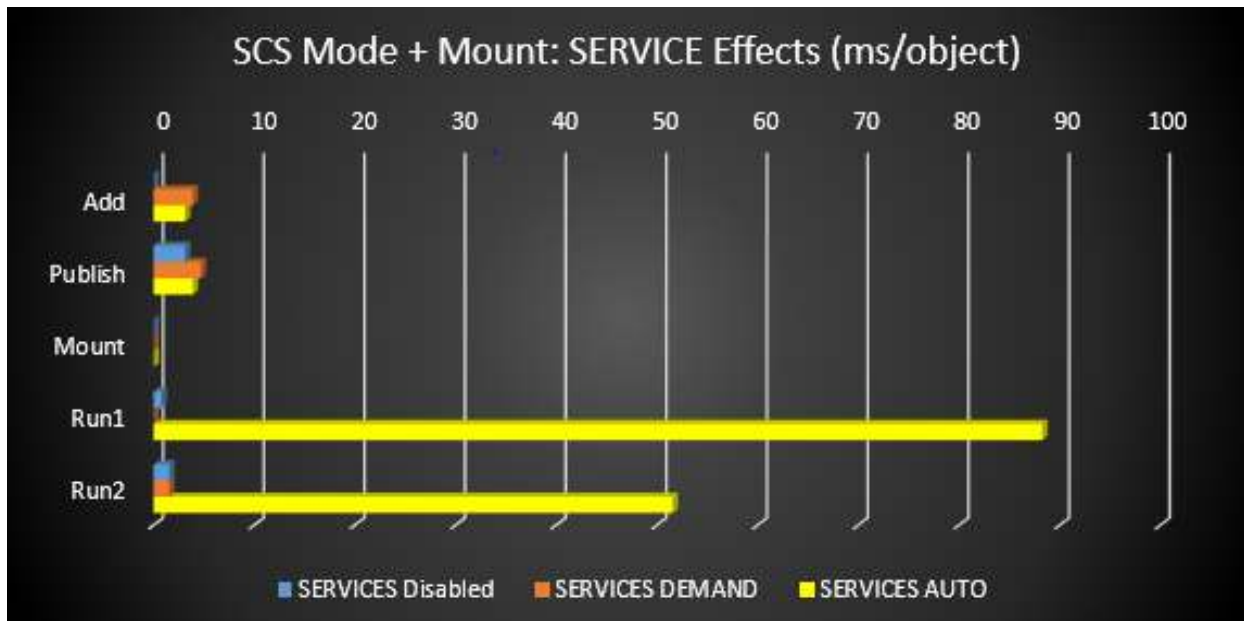
From this (and procmon traces) we determine that the virtual subsystem reads in and processes each of the installed virtual service files at the start of the virtual environment only when the service is set to AutoStart. This happens each time the environment is spun up. Additionally, when more than one virtual service is present, the subsystem appears to process each service serially.

At the end of the virtual environment, any running services are signaled to shutdown in parallel (technically they are signaled serially, but the client doesn't wait for individual services to shut down before signaling the next; the subsystem will then wait for all of them to complete shutdown before releasing the virtual environment).

6.2 SCS Mode Testing with Mounting

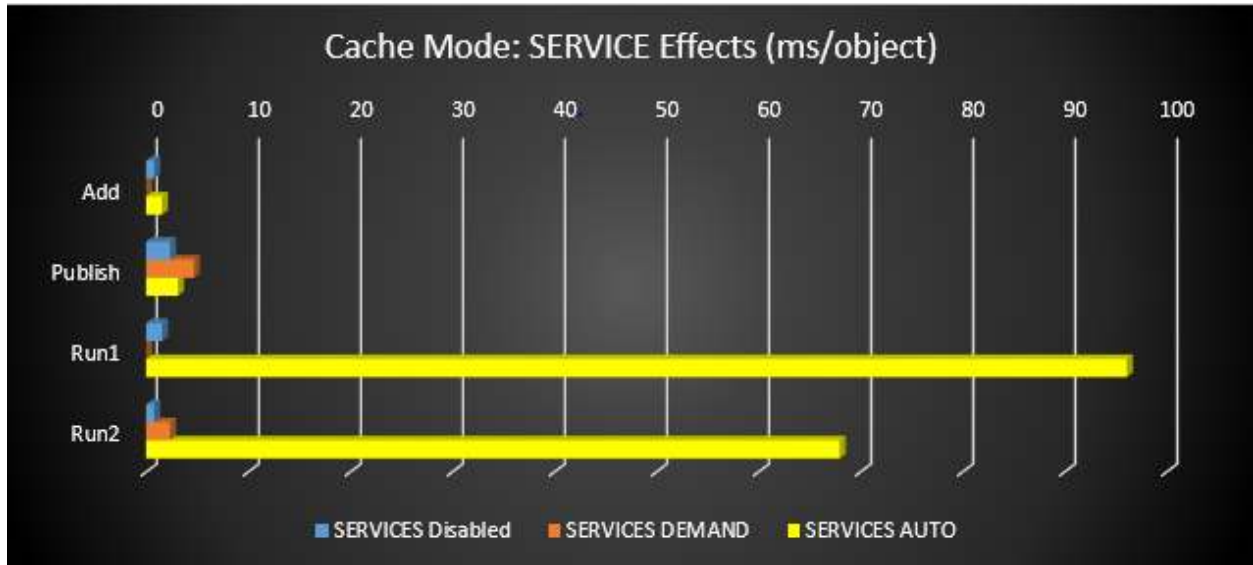
In these tests, SCS Mode is enabled, however, a mount operation is performed prior to running the packages (which performs caching of the package locally, even when SCS mode is enabled). In SCS mode, the autoload setting is ignored (automatically disabled), but mounting may be performed. Although unusual, I wanted to test the possibility of pre-loading certain apps while in SCS mode – which could be done using imaging technology.

The chart which follows shows the results calculated from tests on the various services packages in these tests. The times shown are calculated as the time for the given package minus the time for the BasePkg divided by the number of virtual services, producing a calculated impact in ms/service for each operation.



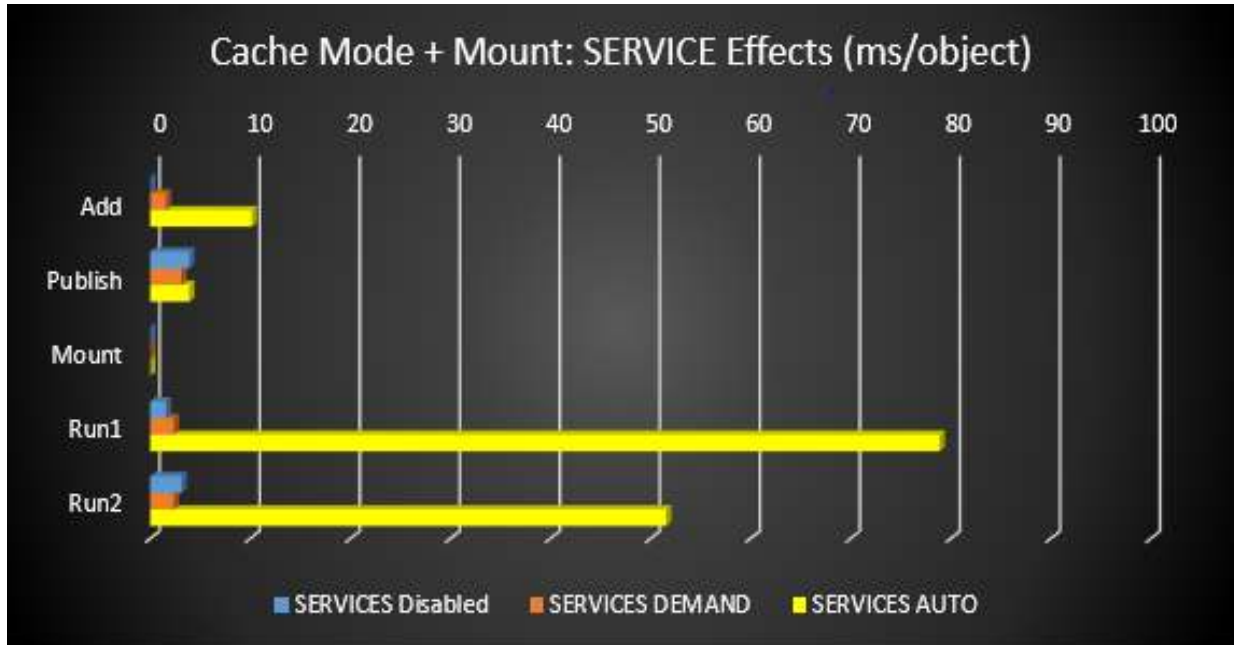
6.3 SCS Mode Disabled, No Mounting

In these tests, SCS Mode is disabled and autoload is also disabled. This is closest to the default setup for clients (where autoload is enabled only for apps previously run) but I completely disabled autoload to keep things simpler.



6.4 SCS Mode Disabled, Mounting Used

In these tests, SCS Mode is disabled and autoload is disabled, but a Mount operation is performed after publishing. This test simulates autoload=2 setting when the user doesn't run the app for a significant period of time after publishing to allow all packages to get fully loaded, but allows for more accurate measurements.



7 About This Research Paper Series

This research paper is part of a series of papers, released by TMurgent Technologies, which investigate the performance impacts that certain application contents can have in the deployment of Microsoft App-V 5 packages.

Through these papers, we can better understand what areas to focus on when packaging applications for App-V when deployment and end-user experience is important. Additionally, with an understanding of these papers you can better target a specific package that is performing poorly and prioritize your efforts to improve it.

TMurgent Technologies, LLP is based in Canton, MA, USA; just 17 miles south of the offices where Microsoft develops the App-V product. TMurgent's Tim Mangan has a long history with the product, having built the original version at Softricity more than a dozen years ago. TMurgent is well known in the App-V community as a source for the best training classes on App-V as well as an endless supply of tools and information. More information is available at the website, www.tmurgent.com