Red Hat Performance Briefs

Optimizing SAS on Red Hat Enterprise Linux (RHEL) 6 & 7

Barry Marson, Principal Software Engineer

Version 1.3.1

April 2019

100 East Davie Street
Raleigh NC 27601 USA
Phone: +1 919 754 4950
Fax: +1 919 800 3804

Send feedback to refarch-feedback@redhat.com

Table of Contents

1 Executive Summary

Today, companies are increasingly utilizing analytics to discover new revenue and cost-saving opportunities. Many business professionals turn to SAS, a leader in business analytics software and service, to help them improve performance and make better decisions faster. Analytics are also being employed in risk management, fraud detection, life sciences, sports, and many more emerging markets. However, to maximize the value to the business, analytics solutions need to be deployed quickly and cost-effectively, while also providing the ability to readily scale without degrading performance. Of course, in today's demanding environments, where budgets are still shrinking and mandates to reduce carbon footprints are growing, the solution must deliver excellent hardware utilization, power efficiency, and return on investment.

To help solve some of these challenges, Red Hat and SAS have collaborated to recommend the best practices for configuring SAS 9 running on Red Hat Enterprise Linux. The scope of this document will cover Red Hat Enterprise Linux 6 and 7. Areas researched include file system selection, the I/O subsystem, and kernel tuning, both in bare metal and virtualized (KVM) environments. Additionally we now include Grid based configurations running with Red Hat Resilient Storage (GFS2 clusters).

This paper discusses:

- **Setup & Tuning Requirements for All SAS Environments**
- **SAS with Red Hat Virtualization**
- **SAS Grid with Red Hat Shared File Systems**
    - **Red Hat Resilient Storage (formerly GFS2 clusters)**
    - **Red Hat Gluster Storage**
    - **Red Hat CEPH Storage**

For information about Red Hat Enterprise Linux 5, see the **Optimizing RHEL for SAS 9.2** document dated January 2011.

2 Setup & Tuning Requirements for All SAS Environments

SAS environments have some very specific setup and tuning requirements to achieve optimal performance.

While there are several new SAS deployments in virtualized (KVM) and GRID based environments, many existing and new installations still require a standalone SAS installation. Its essential to read this chapter regardless of your type of installation.

One of the most important considerations when deploying SAS is choosing the right file system and configuring the I/O stack correctly. Even subtle mistakes can cause serious degradation in performance and scalability. While most performance configuration issues can be resolved, the challenge lies in finding these issues early enough to reduce system downtime. The most common mistakes usually require the recreation of the file system, which means data already on the file system needs to be backed up. To further complicate matters, that archive process can be painfully slow depending on how things were configured. The information here is intended to help you make informed decisions before you deploy.

Areas covered here include:

- **File System Selection**
- **File System Layout**
- **The I/O Stack**
- **Kernel Tuning for SAS**

2.1 File System Selection

Red Hat Enterprise Linux offers several choices of file systems. For stand alone systems, the choice has been pretty clear for several years that the best performing and scaling file systems in order of preference are XFS and ext4. Both of these file systems have extent based allocation schemes which help in their overall file system throughput.

Ext3 simply cannot perform well with the large streaming I/O requirements of many SAS applications mostly due to its non extent based allocation scheme and its indirect block layout. Additionally, ext3 has file and file system size limitations noted in the table below.

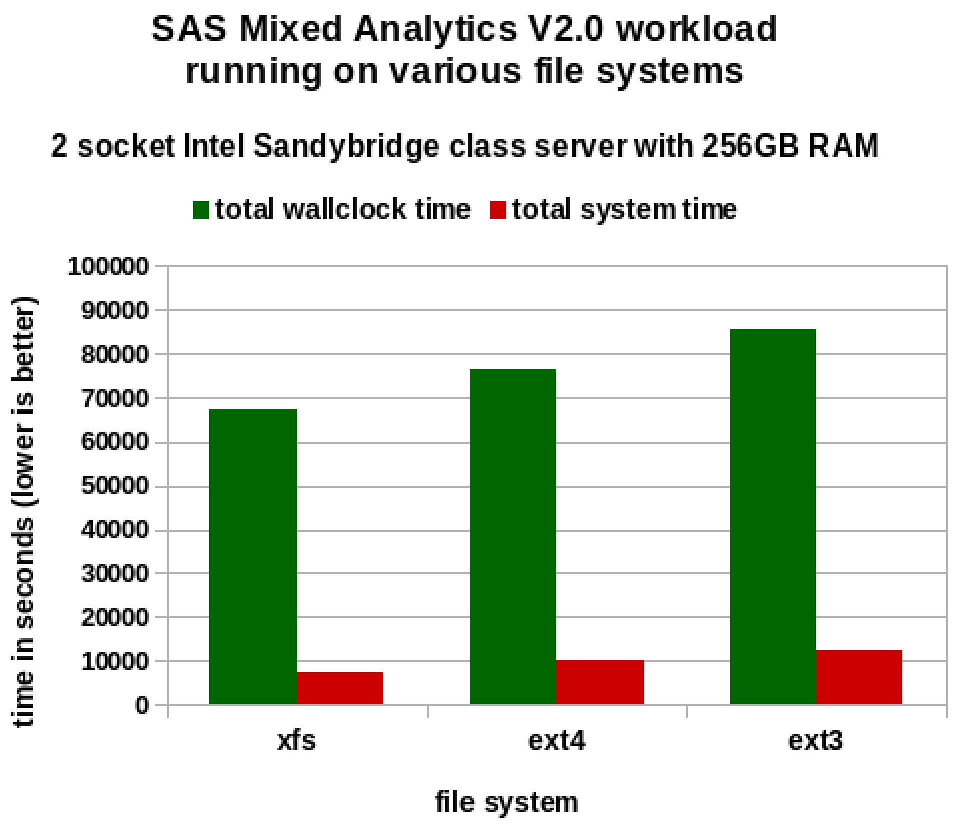
GFS2 clustered file system can be used for SAS Grid Manager and is discussed later in this paper.

btrfs has been dropped in RHEL6 and RHEL7.

Below is a table comparing the file size and file system limits of these four file systems.

	Red Hat Enterprise Linux 6		Red Hat Enterprise Linux 7	
File System Type	**Maximum File System Size**	**Maximum File Size**	**Maximum File System Size**	**Maximum File Size**
XFS	**100 TB**	**100 TB**	**500 TB**	**100 TB**
ext4	**16 TB**	**16 TB**	**50 TB**	**16 TB**
ext3	**16 TB**	**2 TB**	**16 TB**	**2 TB**
gfs2 (shared)	**100 TB**	**100 TB**	**100 TB**	**100 TB**

Below is a graph comparing file system performance when executing a SAS Mixed Analytic V2.0 workload on a simple 2 socket Intel server. The I/O stack for each file system was the same.



total wallclock time - sum of the actual job run times as reported by the SAS log files.
total sysem time - sum of the system CPU time reported by the SAS log files.

2.2 File System Layout

2.2.1 SASDATA

The size of a file system needs to be based on several criteria including the largest file you may need to create. If your largest files approach this file system size limit, you need to ask … “Do I really need to create such a large file ? Can it be broken up into smaller files ?”

You may not want to allocate all the space for a file system up front. You may choose to instead create a smaller file system and extend it’s size later. This is **not** thin provisioning (which is not recommended for SAS environments unless properly discussed with SAS). Recommendations for creating and extending volumes are discussed in the Logical Volume section below.

Both Red Hat and SAS **don't** recommend creating individual file systems as large as their maximum capacity. You need to weigh the need to handle important tasks like backups and maintenance. Monolithic file systems being brought off line would make your entire data set unavailable.

We do highly encourage you to keep each file systems limited to 10-20TB. Of course if you need more space because your data will exceed the file system limits, then you will have to create multiple file systems regardless. But remember, just because a file system can be 100TB in size doesn’t mean if you need 100TB of storage that you should create such a monolithic beast. We highly recommend breaking it up.

If you need larger file systems or aren’t sure how large they should be allowed to grow before breaking them up into multiple file systems, we recommend contacting SAS.

2.2.2 SAS Scratch Space

Due to the nature of how SAS temp file processing works, it is highly recommended that the SASWORK temp space **always** be a separate file system. This is regardless of whether you have a stand alone configuration or a shared file system for GRID. While all SAS file systems have a high I/O demands, this file system typically has the **highest** demand.

2.2.3 SAS Utility File Location

The UTILLOC option specifies one or more locations for a type of utility file that is introduced as part of the multi-threaded procedures in SAS 9 architecture. By default, these files are put in the same directory as where SASWORK points. From a performance perspective, we strongly recommend these files be placed on a separate file system from SASWORK when using HDD spinning drives as it also has high I/O demands.

2.3 The I/O Stack

The I/O Stack effectively covers the block device underneath the file system all the way down to the presented LUN from the storage. Choosing and configuring your I/O stack is essential for optimum performance. If any one piece of the I/O stack is not configured properly, it can have a significant impact on your performance. Here we will discuss:

- **Volume Management (LVM)**
- **Device Mapper Multipath I/O**

2.3.1 Volume Management (LVM)

Logical Volume Manager (LVM) is used to create and manage LUNs made up of single and multiple block devices. These can be bottom level devices (**`/dev/sdc`**) or device mapper presented devices (discussed below). LVM is very useful when you need to construct a LUN bigger than any individual block device. This provides an easy mechanism for creating much larger file systems.

You may be wondering whether you should bother with LVM especially if you would prefer to present a single LUN from your storage array for each file system. The simple answer is both SAS and Red Hat **highly recommend** that you create **several** LUNs and present them to LVM to build your logical volumes. This is especially important with HDD based storage.

LVM also abstracts your devices with names so you don't have to worry about persistent device naming between reboots.

Note: Red Hat Resilient Storage (GFS2) shared file systems **require** LVM.

2.3.1.1 Creating Logical Volumes

When creating a logical volume to contain your file system, it's best to create each segment with the following criteria **especially** if the file system is expected to grow over time.

- Each segment should be no larger than 5 and 10TB in size
- Each segment should be striped as discussed in **Concatenated vs. Striped Logical Volume Segments**.
- SAS typically recommends at least 8 HDD based LUNs or 1-2 SSD LUNs per segment. Many storage arrays perform better with several smaller presented HDD LUNs vs. a single larger LUN. This is highly dependent on the architecture of the storage. Its best to experiment with your resources first. SAS has worked with and consulted on many of the latest storage arrays and can offer the best advice for SAS workloads.

The **`lvcreate(8)`** command is used to create the first striped segment.

2.3.1.2 Increasing the Size of a Logical Volume

When there is a need to grow a logical volume to increase file system capacity, you can accomplish this by creating an additional striped segment and appending it to the end of an existing logical volume. Some IT administrators assume that just adding individual LUNs in a concatenated manner is all that is needed. To maintain high and balanced I/O performance, create each additional striped segments identical in LUN count and stripe settings as the first segment. This makes sure balanced striped performance is achieved on each segment assuming the underlying storage performs uniformly.

Use the **`lvextend(8)`** command to create each new striped segment for the logical volume. LVM will concatenate it to the end of the existing logical volume.

Note: Carefully follow the instructions for growing file systems by consulting the administrators guide.

2.3.1.3 Concatenated vs. Striped Logical Volume Segments

When you construct a logical volume segment from a volume group containing multiple LUNs, by **default**, LVM takes each successive LUN and appends it logically to the end of the prior one. This is known as a **concatenated** volume. This behavior makes it easier to remove a LUN from LVM but can be a serious performance bottleneck. The reason is if there isn't enough parallel activity across the file system then only certain LUNs will be performing I/O while others remain idle. If your volume has been created already, the easiest way to see if it's a concatenated volume is to run:

```
# lvs -o name,vg_name,size,attr,lv_size,stripes,stripesize
LV              VG       Attr        LSize    #Str  Stripe
lroot           vgRHEL6  -wi-ao----  86.71g   1     0
lswap           vgRHEL6  -wi-ao----  8.79g    1     0
```

In the above example the volume group shows a stripe count (**`#Str`**) of 1. That means even if you have multiple LUNs in your volume, they are **not** striped.

To get all your LUNs participating in parallel, tell LVM to create a **striped** volume segment. This stripes blocks across multiple LUNs (like hardware RAID0) to maximize and balance LUN utilization. This ensures, especially with streaming I/O that all LUNs will participate and dramatically improve performance.

Below is an example of striped volumes.

```
# lvs -o name,vg_name,size,attr,lv_size,stripes,stripesize
LV              VG       Attr        LSize    #Str  Stripe
lASUITEinput    vg_sas   -wi-ao----  150.00g  8     64.00k
lASUITEoutput   vg_sas   -wi-ao----  150.00g  8     64.00k
lASUITEsaswork  vg_sas   -wi-ao----  150.00g  8     64.00k
```

The value in the **`Stripe`** column is the amount of data per LUN before I/O moves to the next LUN. The **`#Str`** tells us how many LUNs make up the logical volume. The effective stripe size is (**`#Str x Stripe`**). In the above case 8 x 64KB = 512MB I/O request would request I/O in parallel from all the LUNs. While this typically improves performance even when the LUNs are from the same storage array, the huge win is when LUNS from multiple arrays are configured.

2.3.1.3.1 Arguments for Creating Striped Logical Volume Segments

Below are two important **`lvcreate(8)`** and **`lvextend(8)`** options needed to create a striped logical volume segment. Failure to use these arguments while creating a logical volume segment that is concatenated, not striped.

`-i --stripes Stripes`
: Gives the number of stripes. This is equal to the number of physical volumes to scatter the logical volume.

`-I --stripesize StripeSize`
: Gives the number of kilobytes for the granularity of the stripes.

By default, the **order** with which disks are built into LVM stripes is based on the order they were added into the LVM volume group. If you want to create a stripe in a specific order then you must **specify** that order with the **`lvcreate(8)`** or **`lvextend(8)`** commands. A recommendation is if you plan to stripe with multiple storage arrays, then add a LUN round robin from each of the arrays.

The only drawback of using a striped segment is you cannot migrate data off a subset of LUNs in the segment to free up space. You would have to migrate your data off the entire segment and then remove the segment. We think the performance benefits of LVM striping outweigh this loss of flexibility.

2.3.1.4 LVM Read Ahead

SAS file systems need a significantly elevated read ahead value. This is due to the enormous amount of sequential data being read from files. When LVM creates a logical volume, it sets read ahead to automatic. For SAS environments, it's best to set read ahead to a static large value.

To tune read ahead, include this option with the **`lvcreate(8)`** command or if it was left out, you can change it later with **`lvchange(8)`** :

`-r --readahead {ReadAheadValue|auto|none}`
: Sets the readahead mode for the given logical volume. The default mode is **`auto`**. When **`ReadAheadValue`** is specified the read ahead value becomes static. The default unit is in sectors (512 bytes) but other units like k (kilobytes), m (megabytes) can be used. This value is persistent between reboots.

Read ahead for SAS environments is discussed in depth in the **Additional Tuning Requirements** section.

2.3.2 Device Mapper Multipath I/O

If **either** your server or storage array has multiple connections/paths (Host Bus Adapters [HBA's] or network devices) to a switch, then your system is a candidate for configuring multipath I/O. We **highly** recommend you configure device mapper multipath I/O (MPIO) to provide fail-over and load balancing support. When properly configured, this provides:

- Continuous availability in case a path to a device fails
- Increased bandwidth support between the server and block device
- Easier manageability by providing a single abstracted device path to a presented LUN
- Having multiple and distributed paths to your storage, especially on larger socket systems, additionally helps reduce some latencies associated with Non Uniform Memory Architecture (NUMA) based systems

MPIO requires the **`device-mapper-multipath`** package which is typically not installed by default on your system. Additionally, MPIO will not start up unless the **`/etc/multipath.conf`** file exists at boot up time. Once both of these criteria are met, you will need to reboot your system.

The MPIO driver already has configuration information for several storage arrays. Sometimes configuration file additions and changes need to be made. The easiest way to see the present configuration is to run

```
# multipathd show config
```

This outputs everything the MPIO driver knows about. You can also run the **`multipathd`** command interactively as shown below.

```
# multipathd -k “”
multipathd> show config
```

To view your multipath devices, you can use the command:

```
# multipath -ll
...
msa13_6 (3600c0ff000db3c10c0f5d34d01000000) dm-7 HP,P2000 G3 FC
size=272G features='1 queue_if_no_path' hwhandler='0' wp=rw
   |-+- policy='round-robin 0' prio=130 status=active
   | |- 2:0:0:6 sdj    8:144 active ready running
   | |- 1:0:0:6 sddb  70:144 active ready running
   | |- 1:0:3:6 sdel 128:208 active ready running
   | `- 2:0:7:6 sdcp  69:208 active ready running
   `-+- policy='round-robin 0' prio=10 status=enabled
   |- 2:0:2:6 sdah   66:16  active ready running
   |- 2:0:3:6 sdat   66:208 active ready running
   |- 1:0:2:6 sddz  128:16  active ready running
   `- 1:0:7:6 sdgh  131:208 active ready running
...
```

The above snippet example shows a multipath device with 8 paths that are active, 4 of which will actively handle I/O with another 4 paths enabled and ready for fail-over.

2.3.2.1 Device Mapper Multipath Naming

If you look at the above multipath example, you will notice the device name **`msa13_6`** has been assigned as opposed to something like **`mpath[a-z]`**. This was accomplished by defining an alias name in the **`/etc/multipath.conf`** file. Alias definitions allow you to assign logical names to WWID for easier tracking. This is extremely useful for Resilient Storage configurations since you are not guaranteed the same logical name on each node. It also make it a lot easier to define specific LVM stripes for optimal performance. Below is the config file definition for the above alias.

```
multipaths {
    multipath {
        wwid 3600c0ff000db3c10c0f5d34d01000000
        alias msa13_6
    }
    ...
}
```

2.4 Kernel Tuning for SAS

Having SAS run optimally out of the box is an ultimate desire from both SAS and Red Hat's perspective. We aren't quite there yet but as Red Hat Enterprise Linux has evolved and grown, better operating system defaults and tools help achieve optimum performance easier than ever. The primary areas of the kernel needing tuning for SAS environments include:

- CPU performance and power management
- I/O elevators for enterprise workloads
- Virtual Memory subsystem
- File system read ahead for improved sequential read performance
- File system I/O barriers for data integrity during a system crash

In this section, we will discuss how to tune these areas by discussing:

- **Tuned with Red Hat Enterprise Linux 6 & 7**
- **Creating a Custom Tuned Profile**
- **Additional Tuning Requirements**
- **Tuning Beyond Tuned**
- **SAS Tuning File Examples**

Work began in RHEL5 which provided the **`ktune`** service to provide a single means for adjusting a group of tunables. In RHEL6 we introduced **`tuned`** which provides even more tuning capability. Unfortunately **`tuned`** doesn't completely tune a RHEL system for SAS but is an excellent way to tune most of what SAS needs with a single command.

2.4.1 Tuned for Red Hat Enterprise Linux 6 (RHEL 6)

In RHEL 6, Red Hat extended the **`ktune`** utility to include the **`tuned`** tool which can tune various system parameters based off a profile definition.

For RHEL 6, the most recommended performance profile for file system I/O environments is **`enterprise-storage.`** This profile tunes:

- Adjusts the I/O elevator to **`deadline`** (versus CFQ default)
- Alters the power save mode from OnDemand to Performance
- Sets the VM reclaim parameters for **`dirty_ratio`** back to the RHEL 5 value of 40 (RHEL 6 adjusted default to 20)
- Scheduler tunable quantum back to RHEL 5 default of 10 milliseconds (RHEL 6). Default quantum is 4 milliseconds

- Enables Transparent Huge Pages (discussed below)
- Block device and LVM read ahead values increased by a factor of 4
- Remounts the file systems to disable I/O barriers using “**`-o barrier=0`**” (assumes enterprise class storage with battery backed up controllers). See **`/proc/mount`** to view the barrier settings on the server.

  Early versions of RHEL 6, specifically versions earlier than RHEL 6.2 had improved performance when I/O barriers could be disabled. Significant improvements have been made to barrier code and the difference in performance is typically noise.

The **`tuned`** tool is **not** installed by default in RHEL6 so you will need to install the following rpms:

```
tuned-*.rpm
tuned-utils-*.rpm
```

Useful **`tuned`** commands include:

- To **list** available profiles

```
# tuned-adm list
Available profiles:
- throughput-performance
- desktop-powersave
- sap
- server-powersave
- virtual-guest
- default
- virtual-host
- laptop-battery-powersave
- spindown-disk
- latency-performance
- laptop-ac-powersave
- enterprise-storage
Current active profile: default
```

- To **set** a specific profile

```
# tuned-adm profile enterprise-storage
Stopping tuned: [  OK  ]
Switching to profile 'enterprise-storage'
Applying deadline elevator: dm-0 dm-1 dm-2 sda sdb sdc sdd ... [ OK ]
Applying ktune sysctl settings:
/etc/ktune.d/tunedadm.conf: [  OK  ]
Calling '/etc/ktune.d/tunedadm.sh start': [  OK  ]
Applying sysctl settings from /etc/sysctl.conf
Starting tuned: [  OK  ]
```

- To **show** the active profile

```
# tuned-adm active
Current active profile:enterprise-storage
Service tuned: enabled, running
Service ktune: enabled, running
```

2.4.2 Tuned for Red Hat Enterprise Linux 7 (RHEL 7)

For RHEL 7, the **`tuned`** package is installed by default and for Server variant installs, is set to the **`throughput-performance`** profile by default. This profile effectively replaces **`enterprise-storage`** which was available in RHEL6. The tune is effectively the same with two distinct differences.

1. I/O barriers are not disabled with **`throughput-performance`**. The operating system queries the storage and disables barriers if the storage says it's safe to.
2. The default I/O elevator is now **`deadline`** and is not modified by **`tuned`**.

Note, when you change **`tuned`** profiles in RHEL 7, the tool does not output anything unlike RHEL 6. Use

```
# tuned-adm active
Current active profile: throughput-performance
```

to verify the tune has been enabled.

2.4.3 Creating a Custom Tuned Profile

One question which always comes up is what is the best way to handle the additional tuning requirements needed for SAS environments. The easiest solution is to create a custom **`tuned`** profile. We **highly recommend** this as opposed to modifying an existing profile. The reason is any file installed by RHEL has the potential to be changed from regular updates or errata. Creating your own profile protects you from such updates.

The tuning configuration files for RHEL 6 and RHEL 7 are different and are stored in different directory areas, but the principles for creating a custom tune and tweaking are the same.

RHEL 6

1. Go to the tuning profiles directory at **`/etc/tune-profiles`**. There are sub-directories for each **`tuned`** profile name.

2. Create a new directory that is the name of the profile you want to create. A typical name might be **`sas-performance`**.

3. Copy the files from the **`enterprise-storage`** profile directory into the new **`sas-performance`** directory.

```
# cp /etc/tune-profiles/enterprise-storage/* /etc/tune-profiles/sas-performance
```

4. Edit new file(s) as needed (discussed in the sections below)

5. Enable new profile

```
# tuned-adm profile sas-performance
```

RHEL 7

1. Go to the tuning profiles directory at **`/usr/lib/tuned/profiles`**. There are sub-directories for each **`tuned`** profile name.

2. Create a new directory that is the name of the profile you want to create. A typical name might be **`sas-performance`**.

3. Use **`throughput-performance`** as the basis for your custom profile

   a) RHEL 7 supports class based profiles so you can inherit the characteristics of another profile (with an include statement) and then customization it further

   b) Create a new file in the **`sas-performance`** directory called **`tuned.conf`**

   c) The beginning of the file should include the **`throughput-performance`** profile with

   ```
   [main]
   include=throughput-performance
   summary=Optimize for SAS Applications
   ```

   The summary directive is strictly for documentation purposes when you run **`tuned-adm list`**

4. Any changes made beyond the include will override the **`throughput-performance`** profile

5. Edit new config file as needed (discussed in the sections below)

6. Enable new profile

   ```
   # tuned-adm profile sas-performance
   ```

2.4.4 Additional Tuning Requirements

Below are areas which need further tuning.

- **Flash Storage (I/O elevators and more)**
- **I/O Barriers (RHEL 6 only)**
- **File system read ahead**
- **Virtual Memory Dirty Page Tuning for SAS 9**
- **Transparent Huge Pages (THP)**
- **CPU Power Management for Baremetal Systems**

Note: Before you make any changes to a tuned profile, make sure it is not enabled by setting to another profile.

2.4.4.1 Flash Storage (I/O Elevators and more)

Typically we recommend the default I/O elevator (scheduler) be set to **`deadline`** for all storage devices making up SAS file systems. In RHEL 6, this tuning was handled directly by **`tuned`**. In RHEL 7, the default I/O elevator for server configurations is **`deadline`**. This was and **still** is the recommended setting for HDD **spinning disk media**.

More and more SAS deployments are including **flash** based storage. This type of media typically performs best when their drivers/controllers take full control of I/O scheduling. Such devices should have their I/O elevators set to **`noop`**. The best way to tune I/O elevators in a mixed storage environment is to use a **`udev`** rule.

Below is an example of a **`udev`** rule file from a flash storage vendor.

```
 # Use noop elevator for high-performance solid-state storage
ACTION=="add|change", SUBSYSTEM=="block",ENV{ID_VENDOR}=="PURE",ATTR{queue/scheduler}="noop"
# Reduce CPU overhead due to entropy collection
ACTION=="add|change", SUBSYSTEM=="block",ENV{ID_VENDOR}=="PURE",ATTR{queue/add_random}="0"
# Schedule I/O on the core that initiated the process
ACTION=="add|change", SUBSYSTEM=="block",ENV{ID_VENDOR}=="PURE",ATTR{queue/rq_affinity}="2"
```

`udev` rule files are stored in **`/etc/udev/rules.d`**. Naming conventions must be **`NN-name.rules`** where **`NN`** is a sorting number and **`name`** is a name associating what the rules in the file do. Rules are read in sorted order. The above example might be **`01-purestorage.rules`**.

RHEL 6

Since RHEL 6 **`tuned`** modifies I/O elevators, You need to understand when resources actually get tuned. **`udev`** rules are executed before **`tuned`** starts up during boot-up. This means **`tuned`** will wipe out some of the **`udev`** tuning. To solve this, it is recommended to rerun the **`udev`** rules after **`tuned`** has been brought up.

A way to get the **`udev`** rules rerun is to add the following calls in the **`start()`** function for your profile just before the return call.

```
udevadm control --reload-rules; udevadm trigger
```

A commented out example is shown in the RHEL 6 **SAS Tuning Example** further in this document.

RHEL 7

Since tuned in RHEL 7 does not touch I/O elevators, simply define a **`udev`** rule.

A word of caution when defining udev rules …

*Both Red Hat and SAS recommend **udev** rules only change characteristics about devices that are directly related to device behavior. That is I/O elevators, interrupt pinning, etc. Tuning such as **readahead** should be removed from **udev** rules since **LVM** should handle them.*

2.4.4.2 I/O Barriers (RHEL 6 only)

In RHEL 6, the **`tuned`** profile **`enterprise-storage`** specifically re-mounts all file systems to disable block layer write barriers. Write barriers ensure that certain I/O's make it through the device cache and are on persistent storage. If barriers are disabled on a device with a volatile (non battery backed) write back cache, file system corruption could occur during a crash or power loss.

If you are not sure whether you have battery backed protection then simply remove the calls in **`/etc/tune-profiles/sas-performance/ktune.sh`** which control it.

- In the **`start()`** function, remove

```
disable_disk_barriers
```

- In the **`stop()`** function, remove

```
enable_disk_barriers
```

2.4.4.3 File System Read Ahead

SAS file systems need a significantly elevated read ahead tuning value. This is due to the enormous amount of sequential data being read from files.

The block device that is actually mounted to present the file system is the **only** device which needs to be tuned. Tuning any block device underneath that mounted device will do nothing.

There are several ways to tune read ahead and it's important to set this up correctly or elevated read ahead may not occur or the read ahead rate could be altered automatically. Below are the most relevant block device types.

- **Logical Volume (LVM) Block Devices**
- **Simple or Multipath Block Devices**
- **Third Party Volume Manager Devices**

2.4.4.3.1 Read Ahead for Logical Volume (LVM) Devices

Earlier, we discussed the use of LVM to build and manage powerful striped block devices to house the various SAS file systems. When an LVM block device is configured, it is imperative to **only** use LVM to set read ahead on that block device. Using other mechanisms can conflict with LVM depending on who is setting the read ahead tune at a given time. The following mechanisms, which can tune read ahead, must **not** be used.

1) **Tuned**

To disable read ahead in your **`sas-performance tuned`** profile:

RHEL 6

- Edit the file **`/etc/tune-profiles/sas-performance/ktune.sh`**
- The following line from the shell function **`start()`** should be removed

```
multiply_disk_readahead 4
```

- The following line from the shell function **`stop()`** should be removed

```
restore_disk_readahead
```

RHEL 7

- Edit the file **`/usr/lib/tuned/sas-performance/tuned.conf`**
- Add a **`disk`** subsystem entry with the following entry

```
[disk]
devices=!dm-*
```

- This will prevent **`tuned`** from tuning read ahead for any device mapper device.

2) **udev** rules

   Make sure no special **`udev`** rules define read ahead

3) **Tuning with `/sys/block/*/queue/read_ahead_kb`**

   Make sure you don't tune any device here either manually or with a script

By default, when LVM creates a logical volume, it sets the read ahead to automatic (**`auto`**). For SAS environments, we recommend you set read ahead **manually** to a large static value. The default unit value is in 512 byte sectors. You can specify a different unit as noted below. SAS typically sets read ahead in excess of 8 MB (8192).

To tune read ahead, include this option with your **`lvcreate(8)`** command or if it was left out, you can change it later with **`lvchange(8)`** :

`-r --readahead {ReadAheadValue|auto|none}`

Sets the readahead mode for the given logical volume. The default mode is **`auto`**. When **`ReadAheadValue`** is specified the read ahead value becomes static. The default unit is in sectors (512 bytes) but other units like **k** (kilobytes), **m** (megabytes) can be used. This value is persistent between reboots.

For example to change the read ahead value of an already existing LVM volume to 8MB

```
# lvchange --readahead 8m /dev/mapper/vg_sas-lSASDATA
```

To view the read ahead and other relevant configuration for LVM devices use:

```
# lvs -o name,vg_name,size,attr,lv_size,stripes,stripesize,lv_read_ahead
LV        VG             LSize    Attr        LSize    #Str  Stripe  Rahead
lroot     perf82_R7      272.66g  -wi-ao----  272.66g    1        0    auto
lswap     perf82_R7        4.00g  -wi-ao----    4.00g    1        0    auto
lSASDATA  vg_sas           3.00t  -wi-ao----    3.00t    4  256.00k   8.00m
lSASWORK  vg_sas           3.00t  -wi-ao----    3.00t    4  256.00k   8.00m
lUTILLOC  vg_sas           2.00t  -wi-ao----    2.00t    4  256.00k   8.00m
```

Note that the **Rahead** column in **red** shows explicit read ahead tuning for the SAS logical volumes.

2.4.4.3.2 Read Ahead for Simple or Multipath Block Devices

It is possible to build a SAS file system directly on top of a simple or multipath block device. This is typically **discouraged** for bare metal and most virtualized configurations for performance reasons.

There are situations though, when it can be useful in virtualized environments where the host has an already created striped LVM block device that is passed into the guest. This configuration is typically the exception rather than the norm. This tuning is discussed in the Virtualization section.

2.4.4.3.3 Read Ahead for Third Party Volume Manager Devices

If you are using a third party volume manager, then consult their documentation regarding setting persistent read ahead. In such a case you follow **all** the tuning and setup instructions in the LVM section above that are not LVM specific. The volume manager must be the only tuner for read ahead.

2.4.4.4 Virtual Memory Dirty Page Tuning for SAS 9

Recent analysis of SAS 9 running in customer environments has led to further characterization characterization and demonstrated the need to tune the behavior of dirty page flushing from the page cache. Dirty pages are file system data written in the page cache that have yet to be committed to stable storage. Below are the three primary tuning knobs of interest:

`vm.dirty_background_ratio:`	The percentage of total free and reclaimable memory (which includes the system page cache) at which the background kernel flusher threads will **start** writing out dirty data.
`vm.dirty_ratio:`	The percentage of total free and reclaimable memory (which includes the system page cache) at which a process which is generating disk writes will **itself** start writing out dirty data.
`vm.dirty_writeback_centisecs:`	The kernel flusher threads will periodically wake up and write `old' data out to disk. This tunable expresses the interval between those wakeups,

The tuned profiles we have recommended you base your custom tuned profile on, **`vm.dirty_background_ratio`** is set to 10 and **`vm.dirty_ratio`** is set to 40 (or 30 for virtual-guest).   **`vm.dirty_writeback_centisecs`** by default is set to 500 (or 5 seconds). We will not change this value, but wanted you to be aware of its existence.

2.4.4.4.1 Concerns With Default Values

The problem with **`vm.dirty_background_ratio`** is **`vm.dirty_ratio`** is many systems today have memory systems that have grown in size so fast that flushing data to storage can create huge delays and spikes in I/O activity. Even with solid state storage, sometimes, getting the data to the array managing that storage can simply be overwhelmed.

For example, a system with 256GB of RAM would not start flushing dirty pages to stable storage, in the background, until 10%, or roughly 25GB of qualifying memory (free and reclaimable) was dirty. If the rate you can flush to storage is 2GB/sec, it would take ~12 seconds to flush the data. Additionally, up to 40%, or roughly 100GB of qualifying memory is allowed to be dirty.

The GFS2 file system has even more flushing requirements that effect performance and responsiveness. Each file system has a journal log flush event that triggers every 30 seconds (by default). During this journal log flush, the page cache for that file system is locked for writes and **all** dirty pages associated with that file system are completely flushed to stable storage. In the above example, if a journal log flush event occurred when roughly 10% of of the page cache is dirty, it would block processes from writing to the file system for roughly 12 seconds.

2.4.4.4.2 Tuning Recommendation

To smooth flushing behavior, initiate dirty page flushing earlier, and reduce the time the page cache for a file system is locked (GFS2 specifically), we recommend tuning these value to a more appropriate percentages. Set **`vm.dirty_background_ratio`** to roughly the percentage of memory that can flush all that data (not in theory, but in reality) in 1-2 seconds or roughly

```
    vm.dirty_background_ratio    = 100 * Actual_IO_Rate / Total_Memory

                                 where:   Actual_IO_Rate is in GB/sec
                                          Total_Memory is in GB
```

Round the number up to the next integer percent. In my example above with 256 GB RAM, we were able to flush at 2GB/sec. At that rate,

```
                                 = 100 * 2 / 256 = .78         which rounded up is 1
```

Next, set **`vm.dirty_ratio`** to roughly 5 times that or for my example, 5 (for 5%).

*Note: It is important that **`vm.dirty_background_ratio`** is not set to 0 and is always less than **`vm.dirty_ratio`**.*

To adjust tuning for the new profile **`sas-performance`**

RHEL 6

- Edit the file **`/etc/tune-profiles/sas-performance/sysctl.ktune`**
- Change the **`vm.dirty_ratio`** to the new value
- Uncomment the **`vm.dirty_background_ratio`** line and set to the new value

RHEL 7

- Edit the file **`/usr/lib/tuned/sas-performance/tuned.conf`**
- Add a **`sysctl`** subsystem entry with the following

```
[sysctl]
vm.dirty_background_ratio=N         # where N is the percent desired
vm.dirty_ratio=N                    # where N is the percent desired
```

2.4.4.5 Transparent Huge Pages (THP)

Transparent Huge Pages (**THP**) dynamically migrates standard x86_64 4KB pages to 2MB pages when available. This scanning and conversion is not typically needed in SAS environments when most of memory is devoted to the file system page cache. Both the **`enterprise-storage`** and **`throughput-performance`** profiles enable Transparent Huge Pages.

If you are running an older version of RHEL, specifically RHEL 6.0 or 6.1, then we **highly** recommend you **disable** Transparent Huge Pages. This feature creates much less overhead on SAS performance since RHEL 6.2 and RHEL7 and does not need to be disabled. If you are not sure, It's best to experiment with your workload.

To adjust tuning for a new profile **`sas-performance`**

RHEL 6

- Edit the file **`/etc/tune-profiles/sas-performance/ktune.sh`**
- The shell function **`start()`** contains the following line:

```
set_transparent_hugepages always
```

- Change **always** to **never** to disable Transparent Huge Pages

RHEL 7

- Edit the file **`/usr/lib/tuned/sas-performance/tuned.conf`**
- Add a **`vm`** subsystem with the following entry

```
[vm]
transparent_hugepages=never
```

- The **`never`** value will disable Transparent Huge Pages

When you **enable** this profile, Transparent huge pages will be disabled. You can verify by looking at:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

2.4.4.6 CPU Power Management on Baremetal Systems

Most of the CPU's of systems available for the past decade have the ability to reduce their power footprint by slowing down and even shutting down various subsystems when the system is under light load. The problem is that the performance of some workloads suffer when not a lot of compute cycles are needed. I/O and network loads fall into this category often. The workloads encounter delays due to the time it takes to bring the CPU's fully back online.

To keep a CPU ready to run at a more peak continuous rate, two things must be done.

1. The BIOS in the system needs to be set to **`OS_CONTROL`**. This will give the CPU drivers in the kernel the greatest ability to regulate CPU performance.
2. Modify the custom **`tuned`** configuration to tell the kernel you want the CPUs to remain in a higher power state so they can run faster and provide better responsiveness.

To reduce C-State latency and performance effect, **`tuned`** provides a mechanism to tell the operating system not to allow CPU's to migrate into deeper power management C-States. In the past we have recommended capping the C-State at C1. The problem is newer processors can significantly lose their ability to overclock some CPU cores which can reduce performance.

2.4.4.6.1 Tuning Recommendation

- If you are looking for more predictability at the cost of losing overclocking, then cap at C-State C1.
- To get some overclock feature and have good performance and responsiveness, then cap at C-State C1E which should give you frequency scaling.
- For older processors, we recommend you cap at C-State C3. This seems to be a good sweet spot for many SAS environments.
- Newer processors give you C-States C1, C1E, and C6. For the latest generation of processors we recommend you experiment with difference C-State capping values.

To list the available C-States and it’s corresponding wake up latency cost on an Intel based server with **cpuidle** driver running, you can run the two shell commands below.

```
# cd /sys/devices/system/cpu/cpu0/cpuidle/
# for STATE in state*; do printf "%8s %3d\n" `cat $STATE/name $STATE/latency`; done
```

Some examples from:

Intel(R) Xeon(R) Gold 6142 CPU reports:

```
    POLL   0
  C1-SKX   2
 C1E-SKX  10
  C6-SKX 133
```

To cap at C-State C1E set the value to **10**.

an older Intel(R) Xeon(R) CPU E5-2697A V4 CPU reports:

```
    POLL   0
  C1-BDW   2
 C1E-BDW  10
  C3-BDW  40
  C6-BDW 133
```

To cap at C-State C3 set the value to **40**.

To adjust tuning for the new profile **`sas-performance`**

RHEL 6

- Edit the file **`/etc/tune-profiles/sas-performance/ktune.sh`**
- The shell function **`start()`** contains the following line:

```
        set_cpu_governor performance
```

- **Insert** the following line after it where **`N`** is the C-State capping value:

```
/usr/libexec/tuned/pmqos-static.py cpu_dma_latency=N
```

- The shell function **`stop()`** contains the following line:

```
restore_cpu_governor
```

- **Add** the following line after it:

```
/usr/libexec/tuned/pmqos-static.py disable
```

RHEL 7

- Edit the file **`/usr/lib/tuned/sas-performance/tuned.conf`**
- Add a **`cpu`** subsystem entry with the following

```
[cpu]
force_latency=N                      #   where N is the C-State capping value
governor=performance
energy_perf_bias=performance
min_perf_pct=100
```

This will keep the CPU from going into deep power saving C-States and in certain processors, keep the processor frequency at maximum. Once the profile is enabled, the **`turbostat(8)`** (Intel only) and **`powertop(8)`** tools can be used to monitor processor C-State behavior. Note that some versions of **`turbostat`** require the **`--debug`** flag to view C-State information. Check the manual page for further details as **`turbostat`** features keep evolving.

Below is a RHEL 7 example of **`turbostat`** reporting summary information on an idle system **before** tuning:

```
 # turbostat -S
Avg_MHz %Busy Bzy_MHz TSC_MHz  SMI  CPU%c1  CPU%c3  CPU%c6 CoreTmp Pkg%pc3 Pkg%pc6 time
      5  0.25    1993    2933    0    0.42    4.15   95.18      44    0.00    0.00  5**
      6  0.28    2031    2933    0    0.39    1.01   98.32      45    0.00    0.00  5**
      5  0.27    1993    2933    0    0.73    1.97   97.03      45    0.00    0.00  5**
      ...
```

Note that the average processor speed **Bzy_MHz** on the idle system is a relatively low value frequency and **CPU%c6** tells us that the average state of the cores is over 95% in C-State 6. Of note, on this particular hardware the time it takes to get from C-State 6 to C-State 0 is 200 usec.

Once the **sas-performance** profile is tuned and enabled, rerunning **turbostat** on an idle system will report a higher percentage of time in lower numbered C-States. For this C-State C1 specific example, the average processor speed **Bzy_MHz** at full speed and C-States capped at 1 **CPU%c1**. Of note, on this particular hardware, migrating from C-State 1 to C-State 0 is only 3 usec.

```
 # turbostat -S
Avg_MHz %Busy Bzy_MHz TSC_MHz  SMI  CPU%c1  CPU%c3  CPU%c6 CoreTmp Pkg%pc3 Pkg%pc6 time
      6  0.18    3200    2933    0   99.82    0.00    0.00      48    0.00    0.00  5**
      4  0.13    3200    2933    0   99.87    0.00    0.00      48    0.00    0.00  5**
      4  0.13    3200    2933    0   99.87    0.00    0.00      48    0.00    0.00  5**
      …
```

2.5 Tuning Beyond Tuned

Tuned is a very useful tool for configuring your system for SAS specific needs. If your configuration needs additional tuning, it is best to contact SAS and Red Hat support to discuss your specific needs.

2.6 SAS Tuning File Examples

RHEL 6 `/etc/tune-profiles/sas-performance/ktune.sh`

```
#!/bin/sh

. /etc/tune-profiles/functions

start() {
    # Set CPU and power management to performance
    set_cpu_governor performance
    # IN THIS EXAMPLE WE ARE CAPPING THE CPU C-STATE to C1
    /usr/libexec/tuned/pmqos-static.py cpu_dma_latency=2

    # Disable Transparent Huge Pages
    set_transparent_hugepages never

    # Comment out to Not disable I/O Barriers
    disable_disk_barriers

    # Elevate readahead (disabled)
    # multiply_disk_readahead 4

    # EXAMPLE of rereading udev rules after tuned is finished
    # udevadm control --reload-rules ; udevadm trigger

    return 0
}

stop() {
    # Restore CPU/Power management settings
    restore_cpu_governor
    /usr/libexec/tuned/pmqos-static.py disable

    # Restore Transparent Huge Pages
    restore_transparent_hugepages

    # Comment out to Not disable I/O Barriers
    enable_disk_barriers

    # Reset readahead settings (disabled)
    # restore_disk_readahead

    return 0
}

process $@
```

RHEL 6 `/etc/tune-profiles/sas-performance/sysctl.ktune`

```
# ktune sysctl settings for rhel6 servers, maximizing i/o throughput
#
# Minimal preemption granularity for CPU-bound tasks:
# (default: 1 msec#  (1 + ilog(ncpus)), units: nanoseconds)
kernel.sched_min_granularity_ns = 10000000

# SCHED_OTHER wake-up granularity.
# (default: 1 msec#  (1 + ilog(ncpus)), units: nanoseconds)
#
```

```
# This option delays the preemption effects of decoupled workloads
# and reduces their over-scheduling. Synchronous workloads will still
# have immediate wakeup/sleep latencies.
kernel.sched_wakeup_granularity_ns = 15000000

# If a workload mostly uses anonymous memory and it hits this limit, the entire
# working set is buffered for I/O, and any more write buffering would require
# swapping, so it's time to throttle writes until I/O can catch up.  Workloads
# that mostly use file mappings may be able to use even higher values.
#
# The generator of dirty data starts writeback at this percentage (system default
# is 20%)
#
# SAS 9 TUNING: IN THIS EXAMPLE THESE TWO MODIFIED VALUES FOR vm.dirty_ratio AND
#               vm.dirty_background_ratio ARE TUNED BASED ON A SERVER WITH
#               256GB RAM AND AN I/O SYSTEM THAT CAN WRITE AT 5GB/SEC
#
vm.dirty_ratio = 10

# Start background writeback (via writeback threads) at this percentage (system
# default is 10%)
vm.dirty_background_ratio = 2

# PID allocation wrap value.  When the kernel's next PID value
# reaches this value, it wraps back to a minimum PID value.
# PIDs of value pid_max or larger are not allocated.
#
# A suggested value for pid_max is 1024 * <# of cpu cores/threads in system>
# e.g., a box with 32 cpus, the default of 32768 is reasonable, for 64 cpus,
# 65536, for 4096 cpus, 4194304 (which is the upper limit possible).
#kernel.pid_max = 65536
```

RHEL 7 `/usr/lib/tuned/sas-performance/tuned.conf`

```
# Additional Tuning for SAS

# Profile inherits throughput-performance profile
[main]
include=throughput-performance

[cpu]
# IN THIS EXAMPLE WE ARE CAPPING THE CPU C-STATE to C1
force_latency=2
governor=performance
energy_perf_bias=performance
min_perf_pct=100

[vm]
transparent_hugepages=never

# SAS 9 TUNING: IN THIS EXAMPLE THESE TWO MODIFIED VALUES FOR vm.dirty_ratio AND
#               vm.dirty_background_ratio ARE TUNED BASED ON A SERVER WITH
#               256GB RAM AND AN I/O SYSTEM THAT CAN WRITE AT 5GB/SEC
[sysctl]
vm.dirty_background_ratio=2
vm.dirty_ratio=10

# Disable tuning read ahead on device mapper devices
[disk]
devices=!dm-*
```

3 SAS with Red Hat Virtualization

With the ever increasing need for basic virtualization and cloud computing for provisioning system resources it became important early on to understand the best way to configure those environments for SAS. With respect to basic virtualization, we have characterized SAS performance in KVM environments. Since we had limited hardware resources and the SAS workload requires a large amount of compute, memory,and storage bandwidth, we only focused on a single large virtualized KVM guest.

Areas discussed include:

- **Host Configuration Requirements**
- **SAS Guest Creation**

Make sure you fully read **Setup and Tuning for All SAS Environments** as the information which follows builds on that information.

3.1 Host Configuration Requirements

Regardless of whether you deploy your guests on RHEL 6 or 7 hosts, the basic setup of the host is the same. The I/O and file system requirements are the same as with bare metal environments. The areas specific to virtualized environments which need discussion include:

- **Tuned profile for KVM Host Environment**
- **Kernel Shared Memory (KSM)**

3.1.1 Tuned Profile for KVM Host Environment

The best **`tuned`** profile for the host environment is **`virtual-host`**. This profile assumes that SAS will never be running directly on the host. It focuses on allowing large running processes (KVM guests) and tunes the kernel to favor keeping those processes from paging out when memory resources are low. Additionally it makes sure Transparent Huge Pages (THP) is enabled. This is important because the merging of small **`qemu-kvm`** (KVM) process pages to 2MB pages shows a measurable performance improvement of **10%** when running SAS workloads.

`virtual-host` is a great starting point for tuning key components which are essential for good SAS performance. For peak performance, you will want to create a custom profile which tunes power management components. As noted in the basic setup discussion, you will want to create a custom **`tuned`** profile. We recommend calling it **`sas-virtual-host`**. Follow the instructions above to optimize **CPU power management.** Remember this change will keep your whole host platform running at peak performance for all guests.

3.1.2 Kernel Shared Memory (KSM)

This feature, which is available on RHEL KVM host environments has been shown to only benefit Microsoft KVM guest environments. It should be disabled for SAS on RHEL environments.

RHEL 6

This can be disabled by running:

```
# service ksm stop
# service ksmtuned stop
# chkconfig off ksm
# chkconfig off ksmtuned
```

RHEL 7

This can be disabled by running

```
# systemctl stop ksm.service ksmtuned.service
# systemctl disable ksm.service ksmtuned.service
```

3.2 SAS Guest Creation

The creation of the KVM guests for SAS requires deciding how the resources of your host are going to be divided up. This section will discuss the best practices specific to SAS workloads.

3.2.1 Storage Presentation

One of the most important aspects of configuring virtualized environments for SAS is to get the I/O storage for the SAS specific file systems correct. There are two key ways to present storage to your virtualized guest.

- **PCI pass through devices**

  Passing PCI devices through to the guest yields the least overhead compared to bare metal but provides the least flexibility. Expect overhead of around **2-5%** compared to bare metal. With these devices directly passed into the guest, it is possible to use the Logical Volume manager to build larger block devices out of smaller ones. As discussed in the Basic Setup section, the virtues of striped logical volumes holds true here as well.

- **virtIO**

  If storage is going to be handled via virtIO, then it is **extremely** important to define that storage outside of the KVM guest. For example, if you want to take advantage of LVM and multi-LUN striping, you need to create that LVM block device first. Red Hat Virtualization management tools now provide the functionality to create striped volumes.

  Performance characterization has shown that SAS performs best with the very latest version of RHEL 7 host and KVM guest bits. For several years, virtIO performance running SAS workloads has had a **15%** overhead compared to bare metal. Enhancements in RHEL 7.1 have reduced that overhead to around **7%**.

If virtIO is to be used, the following is highly recommended. For each virtIO block device handling a SAS file system:

- Make sure the block device is a **raw** device not QCOW2. QCOW2 cannot handle the vast I/O requirements of SAS.
- Cache mode should be set to **none** which makes sure data written from the virtualized environment is committed to storage
- I/O mode should be set to **native** not threads.

3.2.2 KVM Guest Memory Allocation

Normally when KVM guests start, the KVM process allocates memory only on demand. SAS guests typically require significant amounts of memory because they like to make use of the page cache. We recommend forcing the guest to pre-allocated memory resources up front. This allows the host to effectively allocate Transparent Huge Pages quickly as well as reduce performance stutter early in the life of the guest.

3.2.3 Pinning KVM Guests

If your KVM guest is small enough to fit in a sub set of NUMA nodes on your host server, consider pinning the guest for better and more predictable performance.

3.2.4 Tuned Profile in a Virtualized Guest

For both RHEL 6 and 7, the **`tuned`** profile **`virtual-guest`** is the best tuning profile to start with. This profile is similar to **`throughput-performance`.** A custom profile is still needed for the environment running SAS. We recommend calling it **`sas-virtual-guest`**. The recommendations in **Additional Tuning Requirements** apply here as well however the following must be understood.

1. I/O barriers should be left **enabled**.

2. CPU performance and power management tuning is not needed and should not be added to the config file. That is handled by the host.

3. If virtIO block devices are simply block devices to be managed by a volume manager (LVM or third party) inside the guest, then volume creation as well as read ahead tuning is setup identically to what is documented for **Setup & Tuning for All SAS Environments** as defined in File System Read Ahead.

4. If virtIO block devices are **actually** LVM logical volumes defined from the host, then read ahead needs to be configured differently. This is because the guest has **no idea** LVM is even involved. In such a case **`tuned`** ***will*** be used to set readahead.

   To adjust tuning for a new profile called **`sas-virtual-guest`**

RHEL 6

- Edit the file **`/etc/tune-profiles/sas-virtual-guest/ktune.sh`**
- The shell function **`start()`** contains the following line:

```
multiply_disk_readahead 4
```

- Tuning readahead in RHEL 6 **`tuned`** is by a multiplier. By default, simple LUN readahead values are 128KB. The **`tuned`** multiplier elevates it 4 times to 512KB. SAS typically wants them to be between **8192 KB** and **16384 KB**
- To elevate them that much, change the multiplier from 4 to 64 or 128.

RHEL 7

- Edit the file **`/usr/lib/tuned/sas-virtual-guest/tuned.conf`**
- Add a **`disk`** subsystem entry with the following entry

```
[disk]
readahead=>8192
```

- The unit is in KB by default.
- SAS typically wants them to be between **8192 KB** and **16384 KB**.
- The **=>** instructs **`tuned`** to elevate a block device if the readahead value is not already above **8192**. If you have a LUN whose default readahead value is already elevated then tuned will not down tune it if its value is already higher than this value.

When you enable this profile, the read ahead values will be elevated. You can inspect readahead values with the **`blockdev(8)`** command. For a list of all block devices use:

```
# blockdev --report
```

or you can look at an individual device with:

```
# blockdev --getra DEVICEPATHNAME
```

4 SAS Grid with Red Hat Shared File Systems

Presently the best solution for entry level sized clusters for SAS Grid is to use Red Hat Resilient Storage (GFS2 clusters). This cluster specifically requires the GFS2 file system as the shared file system. This environment supports 2 to 16 nodes in a cluster. GFS2 can support a 100 TB file system. The maximum file size is 100 TB.

Presently SAS GRID is ***not*** supported with either Red Hat Gluster Storage or Red Hat CEPH Storage environment.

Other shared file systems are supported on Red Hat Enterprise Linux, but are not discussed here. More information can be found on SAS's website:

http://support.sas.com/resources/papers/proceedings17/SAS0569-2017.pdf

4.1 Red Hat Resilient Storage (GFS2 Clusters)

As early as 2011, SAS began working with Red Hat to determine if a SAS Grid environment could be deployed with Red Hat Resilient Storage (formerly known as GFS2 clusters). There were already a few customers using this clustered environment with mixed success. SAS created a Grid workload called Calibration to validate whether the environment could handle the rigors of SAS Grid. This calibration workload is a meta-data intensive Grid workload broken up into 750 SAS jobs. Originally, SAS was unable to get reliable and repeatable performance results. Red Hat performance engineering and SAS worked closely with the Red Hat cluster and GFS2 file system team and after 18 months were able to validate and get SAS's acceptance that Red Hat Resilient Storage performed well.

4.1.1 Supported Versions of Red Hat Enterprise Linux

RHEL 6

Red Hat Resilient Storage 6.4 is the first update version of RHEL 6 which officially supports SAS Grid Manager. While there may have been attempts with customers using the environment in earlier versions of RHEL, neither Red Hat nor SAS recommend it. This is because the changes not only improved stability, but also performance and scalability. Due to the extensive changes in RHEL 6.4, engineering will not back port these fixes to earlier versions of RHEL 6. Additionally, earlier environments attempting to migrate forward to RHEL 6.4 or later should archive and rebuild their file systems. This is because one area heavily addressed was file system fragmentation.

RHEL 6.5 introduced additional GFS2 file system enhancements which include support for the Orlov allocator. Testing showed a **9%** improvement over RHEL 6.4 using the same 4 node cluster.

Please note that all validation on RHEL6 was done with the classic CMAN based cluster product

SAS Grid is now supported on RHEL 7.1 and later. At this time, the stability and performance enhancements required to support SAS have not been back ported to RHEL 7.0.

4.1.2 Cluster Considerations

4.1.2.1 Architecture Review

Due to the complexities of a cluster, Red Hat and SAS require initiating a SAS Grid review as well as highly recommend a RHEL Resilient Storage Architecture review. This is to make sure that the customers functional configuration and performance expectations can be met. Additionally these reviews, along with this guide, help the customer make better decisions about deploying their configuration.

Additional information about the RHEL Resilient Storage Architectural Review can be found at the document: https://access.redhat.com/articles/2359891

4.1.2.2 Stretch Clusters

Both the Cluster Logical Volume daemon (**`clvmd`**) and GFS2 file system are not supported in a stretch cluster configuration and thus stretch clusters cannot be configured for SAS.

4.1.2.3 Number of Nodes

Red Hat Resilient Storage supports configurations between 2 and 16 nodes. Red Hat has run with as many as 6 nodes, SAS has validated with 4, 8 and 12 nodes. Testing has been done with iSCSI as well as fiber channel attached storage.

Red Hat Resilient Storage is an excellent cluster file system for small configurations. It offers great performance and low end scalability when properly configured.

4.1.2.4 Dedicated Cluster Interconnect NIC

Heavy metadata traffic in a clustered environment generates significant cluster interconnect traffic. This traffic is typically in the form a several small network messages. While it is possible to share the network device, it is highly recommended to keep it isolated.

Additionally we highly recommend the cluster interconnect host names be defined in each /etc/hosts file to remove any chance of latencies with DNS look-ups.

4.1.2.5 Excellent I/O Connectivity and Performance

SAS processing puts a huge demand on the I/O subsystem. It is extremely important to make sure you have excellent connectivity from many / all sockets of your servers. When multipath I/O is available it should be utilized.

4.1.2.6 SAS Scratch Space Considerations

Some SAS Grid configurations may want to put their scratch space (SASWORK) scratch space on the same file system as their primary data. This is usually for convenience so the IT group only has to manage persistent data via the shared file system. SASWORK space **must** be on a **separate** file system from SASDATA. One of our validation tests, is to run multiple iterations of SAS Calibration. After each iteration we save the output files on the file system. This file system aging test validates how well the files are being allocated on SASDATA as well as how the file system performs as it gets filled up.

Testing with a separate SASDATA shared file system showed **predictable and repeatable** performance even when the file system was at 85% capacity.

Testing with the SASWORK directory on the SASDATA file system showed that even after the first iteration, high levels of file fragmentation had occurred. This continued to worsen with each iteration. File fragmentation hurts I/O performance because the storage needs to seek to different parts of the storage to find the next extent of data. This can be a huge performance loss especially with spinning disk drives. The kernel has to initiate many more smaller I/O requests regardless of disk type.

4.1.2.6.1 Local File Systems

The only time SAS prefers SASWORK to be part of a shared file system is when a SAS application **requires** it for fail over, recovery or rescheduling. This is done by the SAS user adding checkpoint/restart SAS statements and logic to the SAS job. It is not based on a SAS product.

Customers who seek even more performance at the cost of losing re-animation ability and additional IT management can create local SASWORK file systems on each node. There is no question that a local SASWORK file system (when given the proper I/O resources and configuration) can outperform a shared file system. The local file system should be XFS. Again, you have to weigh the pros and cons.

4.1.2.7 Distributed Lock Manager (DLM) Tuning

RHEL 6

There are three tunables which improve DLM look-up performance. The three hash tables are:

```
DLM_LMBTBL_SIZE
DLM_RSBTL_SIZE
DLM_DIRTBL_SIZE
```

These tunables live in the config file **`/etc/sysconfig/cman`** By default they are 1024 but should be elevated to their maximum (16384).

This tuning must be done **before** the shared file systems are mounted.

RHEL 7

There is no DLM tuning for RHEL 7. The DLM hash table was rewritten for much better scalability.

4.2 Red Hat Gluster Storage (RHGS)

Red Hat's performance engineering team has made several attempts over the years to get the SAS Grid Calibration workload running in a Gluster environment. Each time we get closer, but the environment is still not suitable for the type of file system and I/O calls that SAS issues.

Today, Gluster storage is not an option today for SAS Grid.

4.3 Red Hat CEPH Storage (RHCS)

SAS specifically requires a POSIX compliant file system. Cephfs provides that interface. While cephfs became a supported file system in RHCS 2.1 is not an option today for SAS Grid. More research is needed to see if it can be used for standalone configurations.

Appendix A: Revision History

Revision 1.0	August 28, 2015	Barry Marson

Initial Release

Revision 1.1	March 10, 2016	Barry Marson

Clarification of read ahead tuning especially in virtualized environments, fixed readahead example

Revision 1.2	May 10, 2017	Barry Marson

Various typos and fixed font changes. Updates on file system size and layout, LVM volume creation and growing and RHEL7 support documentation; updated SAS paper links, and architectural review process ptr

Revision 1.3	April 5, 2019	Barry Marson

Added documentation associated with tuning vm page cache dirty pages and updated power management.

Revision 1.3.1	April 22, 2019	Barry Marson

Fix minor edit with transparent_hugepages tuning typo