Table of Contents

Executive Summary .................................................................................................................................................................................................. 3
Introduction ................................................................................................................................................................................................................... 3
Overview of Sparse VMDK formats .................................................................................................................................................................. 4
  VMFSsparse ........................................................................................................................................................................................................... 4
  SEsparse .................................................................................................................................................................................................................. 5
Iometer Workloads .................................................................................................................................................................................................... 5
  Test-Bed Architecture ....................................................................................................................................................................................... 5
  Test Methodology .............................................................................................................................................................................................. 6
  Iometer Workload Configuration ................................................................................................................................................................. 7
Performance Results on Local Storage ..................................................................................................................................................... 7
  Empty VMDK ................................................................................................................................................................................................... 7
  Full VMDK ......................................................................................................................................................................................................... 8
Performance Results on SAN Storage ...................................................................................................................................................... 9
  Empty VMDK .................................................................................................................................................................................................. 9
  Full VMDK ........................................................................................................................................................................................................ 9
Hadoop MapReduce Applications .................................................................................................................................................................... 10
  Test-Bed Architecture ...................................................................................................................................................................................... 11
  Test Environment ............................................................................................................................................................................................ 12
  Results .................................................................................................................................................................................................................... 13
VDI Workload (View Planner) ..................................................................................................................................................................... 14
  Test Scenario and Environment ................................................................................................................................................................. 14
  Results .................................................................................................................................................................................................................... 15
Conclusion ................................................................................................................................................................................................................... 17
References ................................................................................................................................................................................................................... 18
Executive Summary

The SEsparse virtual disk format was introduced in VMware vSphere® 5.1 for VMware® Horizon View™ environments where reclamation of storage space is critical because of the large number of tenants sharing storage. In vSphere 5.5, for VMDKs greater than 2TB in size, SEsparse becomes the default scheme for virtual disk snapshots. Various enhancements were made to SEsparse technology in the vSphere 5.5 release, which makes SEsparse perform mostly on par or better than VMFSsparse formats. SEsparse also has a significant advantage over VMFSsparse virtual disk formats by being space efficient.

We conducted a series of performance experiments, including a comprehensive set of lometer workloads, real data-intensive applications like Hadoop MapReduce applications, and VDI workloads. Overall, the performance of SEsparse is about 2x better than the VMFSsparse format for a random write workload and slightly better or on par with the VMFSsparse format for other workloads. One of the very few cases where VMFSsparse outperforms SEsparse is during sequential writes of very large block sizes like 512KB. The data generation part of the Hadoop TeraSort application issues large (512KB) sequential writes, so we have seen decreased performance in SEsparse for those cases. Improving the sequential write performance with large I/Os is being investigated. For VDI environments, however, using the SEsparse virtual disk format increases the space efficiency of VDI desktops over time with no impact on user latencies. The space reclamation (wipe-shrink) operation in SEsparse has a 10% CPU overhead and should be scheduled during low server load. After the wipe-shrink operation completes, we observe slight improvements in user latency and CPU utilization. Overall, SEsparse is the recommended disk format for VDI workloads.

Introduction

Limited amounts of physical resources can make large-scale virtual infrastructure deployments challenging. Provisioning dedicated storage space to hundreds of virtual machines can become particularly difficult. VMware vSphere 5.5 provides two linked-clone techniques [1], VMFSsparse and SEsparse, which were designed to reduce the storage space requirements in virtual deployments. These techniques allow multiple virtual machines to run off delta-disks sharing the same base parent. When created, the delta-disks take minimal physical space and they grow with every new write I/O operation performed by the virtual machine. Conceptually, if the VMs don’t write any data, the amount of storage needed for them to run would be limited to the space allocated to their parent disk.

Since the I/O characteristics vary significantly between applications, a good understanding of the features as well as the performance of different sparse virtual disk technologies will help system administrators and developers choose the solution tailored best to their applications’ attributes. The first part of this paper presents a high level overview of VMFSsparse and SEsparse, emphasizing a key architectural difference between the two; namely, the ability of SEsparse to dynamically reclaim unused disk space. The second part presents the results of an in-depth performance study of linked-clone technologies. The first set of performance tests were conducted with lometer to focus strictly on the storage performance. For the other two scenarios of the performance study, we look at the real world performance of two cutting edge application domains: Big Data Analytics and Virtual Desktop Infrastructure (VDI). Apache Hadoop and VMware View workloads were selected to represent these domains, respectively.

The performance of SEsparse and VMFSsparse is evaluated through a comprehensive matrix of lometer-generated workloads with different data transfer sizes and I/O access patterns using delta-disks hosted on a SAN and a local disk. The performance of SEsparse and VMFSsparse is also compared again to a thin-provisioned disk as a baseline.

Because Hadoop MapReduce applications use temporary storage to host intermediate results, these applications are perfect candidates with which to employ the space reclamation feature provided by SEsparse. How well such applications perform on delta-disks is also covered in this paper.
Since VMware View uses SEsparse during the creation of linked clones, the performance of View workloads on SEsparse virtual disks in vSphere 5.5 is also studied in this paper. VMware View Planner 3.0 [2] [3] is used to generate the VDI workload in each of the 40 VMs executing on the VMware ESXi™ server. View Planner results measuring client-side latencies of user operations and server-side CPU utilization are discussed. VM tunings to maximize space reclamation through SEsparse are also discussed.

**Overview of Sparse VMDK formats**

The two sparse virtual disk formats are VMFSsparse (redo-logs) and SEsparse. Sparse VMDKs are created during (1) Creation of linked clones and (2) VM snapshotting. The SEsparse format replaced the VMFSsparse format during creation of linked clones as of vSphere 5.1. In vSphere 5.5, the default sparse format created during VM snapshot operations is still VMFSsparse. However, this is true only for VMs with VMDKs less than 2TB in size because that is the maximum size supported by VMFSsparse. vSphere 5.5 supports VMDKs larger than 2TB in size, and a snapshot of a VM with VMDKs bigger than 2TB will use the SEsparse format. This distinction is handled internally and is transparent to the user.

**VMFSsparse**

VMFSsparse is a virtual disk format used when a VM snapshot is taken or when linked clones are created off the VM. VMFSsparse is implemented on top of VMFS and I/Os issued to a snapshot VM are processed by the VMFSsparse layer. VMFSsparse is essentially a redo-log that grows from empty (immediately after a VM snapshot is taken) to the size of its base VMDK (when the entire VMDK is re-written with new data after the VM snapshotting). This redo-log is just another file in the VMFS namespace and upon snapshot creation the base VMDK attached to the VM is changed to the newly created sparse VMDK.

Because VMFSsparse is implemented above the VMFS layer, it maintains its own metadata structures in order to address the data blocks contained in the redo-log. The block size of a redo-log is one sector size (512 bytes). Therefore the granularity of read and write from redo-logs can be as small as one sector. When I/O is issued from a VM snapshot, vSphere determines whether the data resides in the base VMDK (if it was never written after a VM snapshot is taken) or if it resides in the redo-log (if it was written after the VM snapshot operation) and the I/O is serviced from the right place. The I/O performance depends on various factors, such as I/O type (read vs. write), whether the data exists in the redo-log or the base VMDK, snapshot level, redo-log size, and type of base VMDK.

**I/O type:** After a VM snapshot takes place, if a read I/O is issued, it is either serviced by the base VMDK or the redo-log, depending on where the latest data resides. For write I/Os, if it is the first write to the block after the snapshot operation, new blocks are allocated in the redo-log file, and data is written after updating the redo-log metadata about the existence of the data in the redo-log and its physical location. If the write I/O is issued to a block that is already available in the redo-log, then it is re-written with new data.

**Snapshot depth:** When a VM snapshot is created for that first time, the snapshot depth is 1. If another snapshot is created for the same VM, the depth becomes 2 and the base virtual disks for snapshot depth 2 become the sparse virtual disks in snapshot depth 1. As the snapshot depth increases, performance decreases because of the need to traverse multiple levels of metadata information to locate the latest version of a data block.

**I/O access pattern and physical location of data:** The physical location of data is also a significant criterion for snapshot performance. For a sequential I/O access, having the entire data available in a single VMDK file would perform better compared to aggregating data from multiple levels of snapshots such as the base VMDK and the sparse VMDK from one or more levels.

**Base VMDK type:** Base VMDK type impacts the performance of certain I/O operations. After a snapshot, if the base VMDK is thin format [4], and if the VMDK hasn’t fully inflated yet, writes to an unallocated block in the base thin VMDK would lead to two operations (1) allocate and zero the blocks in the base, thin VMDK and (2) allocate and write the actual data in the snapshot VMDK. There will be performance degradation during these relatively rare scenarios.
SEsparse

SEsparse is a new virtual disk format that is similar to VMFSsparse (redo-logs) with some enhancements and new functionality. One of the differences of SEsparse with respect to VMFSsparse is that the block size is 4KB for SEsparse compared to 512 bytes for VMFSsparse. Most of the performance aspects of VMFSsparse discussed above—impact of I/O type, snapshot depth, physical location of data, base VMDK type, etc.—applies to the SEsparse format also.

In addition to a change in the block size, the main distinction of the SEsparse virtual disk format is space-efficiency. With support from VMware Tools running in the guest operating system, blocks that are deleted by the guest file system are marked and commands are issued to the SEsparse layer in the hypervisor to unmap those blocks. This helps to reclaim space allocated by SEsparse once the guest operating system has deleted that data. SEsparse has some optimizations in vSphere 5.5, like coalescing of I/Os, that improves its performance of certain operations compared to VMFSsparse.

Iometer Workloads

In this and the following sections, we present the performance results comparing sparse VMDK formats (VMFSsparse and SEsparse) and the thin VMDK format. We consider these three formats because they are similar in terms of “on-demand” allocation of blocks where they do not occupy space when they are created, and they grow as data is written to them. Note, however, that the metadata information is stored and accessed differently for these three formats.

Test-Bed Architecture

<table>
<thead>
<tr>
<th>LOCAL STORAGE TEST BED</th>
<th>SAN STORAGE TEST BED</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Analyzer VM</td>
<td>I/O Analyzer VM</td>
</tr>
<tr>
<td>lometer</td>
<td>lometer</td>
</tr>
<tr>
<td>Virtualized Host</td>
<td>Virtualized Host</td>
</tr>
<tr>
<td>datastore</td>
<td>datastore</td>
</tr>
<tr>
<td>VMDK</td>
<td>VMDK</td>
</tr>
<tr>
<td>- SEsparse</td>
<td>- SEsparse</td>
</tr>
<tr>
<td>- Redolog</td>
<td>- Redolog</td>
</tr>
<tr>
<td>- Thin</td>
<td>- Thin</td>
</tr>
<tr>
<td>RAID5</td>
<td>SAN</td>
</tr>
</tbody>
</table>
Table 1. Local and SAN storage test-bed configuration and methodology

**Test Methodology**

A single virtual machine running Iometer generated a matrix of workloads covering a wide range of data transfers across different I/O access patterns, as described in Table 1. The delta-disks used in this performance study were created from an eager-zeroed thick parent using the default block size for VMFSsparse and 4KB blocks for SEsparse.

Because a performance difference between empty and full VMDKs is expected, we evaluated both. We call “empty” a newly created sparse VMDK and “full” a sparse VMDK pre-filled with randomly generated data.

In the case of empty VMDKs, each experiment ran on a fresh thin or sparse disk, after the load generator VM was power cycled. In the full case, the entire matrix of experiments ran to completion without any interruptions. Because all read I/O operations from an empty linked-clone VMDK, without any writes in between, are serviced from the parent disk, we considered the 100% reads irrelevant to the overall performance of the empty delta-disk itself, hence we chose to evaluate only the performance of write I/O operations.

**Table 2. Types of reads and writes and block size evaluated for empty and full VMDKs**

---

*Most Hadoop deployments utilize internal drives to host HDFS and temp data; therefore, the baseline performance of a single disk is relevant in the context of Hadoop MapReduce applications.*
Iometer Workload Configuration

<table>
<thead>
<tr>
<th>LOCAL STORAGE TEST BED</th>
<th>SAN STORAGE TEST BED</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 16 outstanding I/O operations</td>
<td>• 32 outstanding I/O operations</td>
</tr>
<tr>
<td>• 1 worker thread</td>
<td>• 1 worker thread</td>
</tr>
<tr>
<td>• 4KB aligned disk accesses</td>
<td>• 4KB aligned disk accesses</td>
</tr>
<tr>
<td>• 300 seconds runtime</td>
<td>• 300 seconds runtime</td>
</tr>
</tbody>
</table>

Table 3. Iometer workload configuration

Performance Results on Local Storage

This study is a preamble to the Hadoop MapReduce performance evaluations, and its objective was to generate an I/O throughput baseline comparison between VMFS sparse, SE sparse, and Thin VMDKs hosted on a VMFS volume spanning a single, internal physical disk. An identical physical disk and VMFS setup was used in the Hadoop cluster deployment for each of the VMDKs holding Hadoop’s HDFS and temporary data.

The graphs below illustrate the performance of VMFS sparse and SE sparse delta-disks compared against the thin VMDK. We collected the results for both the empty and full cases according to the matrix of experiments in Table 2.

Empty VMDK

Figure 1. Write performance of empty VMDKs hosted on a single local disk

On an empty VMDK, SE sparse write performance is higher than the thin and VMFS sparse formats for most I/O sizes with two exceptions: 512KB random write I/O size where VMFS sparse performance is higher and 4KB sequential write I/O size where the thin format has the performance edge. Thin random write performance is the lowest across all the data points because for the thin case, we first zero the blocks before writing actual data. This is because VMFS allocates blocks at a 1MB granularity while only part of that area may be filled with real data. Zeroing prevents applications from reading sensitive residual data from an allocated 1MB region of the physical media. In contrast, when reading from SE sparse and VMFS sparse formats, allocations happen in much smaller block sizes, namely 4KB and 512 bytes, respectively, and therefore there is no need to zero the blocks if I/O is at least 4KB and it is 4KB aligned (for the other cases, we do a read-modify-write operation).

SE sparse performs far better than the thin and VMFS sparse formats in the case of random writes. This is because SE sparse implements intelligent I/O coalescing logic where these random I/Os are coalesced into a bigger I/O and the disk controller does a better job of scheduling these I/Os for better performance. Note that SE sparse...
performs on par with or better than VMFSsparse only in cases when the I/Os are aligned to 4KB boundaries. This is because for I/Os smaller than 4KB, or if an I/O is not aligned to the 4KB boundary, writes to SEsparse can result in a read-modify-write operation, increasing overhead. However, almost all file systems and applications are 4KB aligned and therefore SEsparse performs well in common use cases.

Full VMDK

Figure 2. Read/write performance of “full” VMDKs hosted on local storage across different data transfer sizes

Figure 2 shows the performance of sparse virtual disk formats compared to the thin VMDK format. It is clear that the thin format outperforms sparse formats for random accesses. This is because the thin VMDK format maintains and accesses only a small amount of metadata compared to sparse formats. For sparse formats, there is base VMDK metadata and then separate metadata for sparse VMDKs. For every I/O, these additional metadata structures are consulted when servicing the I/O. Thin performance maps very closely to SEsparse for sequential.

Comparing the SEsparse and VMFSsparse results clearly shows that the random performance of SEsparse is consistently better for both reads and writes. While sequential read performances of SEsparse and VMFSsparse are almost on par, sequential write performance of SEsparse is better by a significant margin. This is because I/Os to VMFSsparse are issued in synchronous mode by the virtual SCSI controller, and the guest virtual CPU is blocked until the I/O reaches the physical device driver. For SEsparse, on the other hand, the I/Os are issued asynchronously. Therefore the number of outstanding I/Os from the virtual SCSI layer for VMFSsparse will be
much less compared to SEsparse. Because the physical disk drive is located below a RAID controller with a memory cache, the response time for the device is very low. VMFSsparse does not utilize the low-latency device adequately because of issuing synchronous requests.

**Performance Results on SAN Storage**

The following results compare the performance of the sparse virtual disk formats with the thin VMDK format when using a SAN array–based storage. In this case, the VNX 5700 storage array is connected to the host through a Fibre Channel interface. The goal of these experiments is to illustrate how SEsparse and VMFSsparse perform when the underlying storage device is higher performing than a local disk.

**Empty VMDK**

![Sequential Writes (on empty VMDK)](image1)

![Random Writes (on empty VMDK)](image2)

Figure 3. Write performance of “empty” VMDKs hosted on SAN storage across different data transfer sizes

**Full VMDK**

In this case, we compare thin and sparse virtual disk formats when the VMDK is fully filled. This means all the blocks are already allocated before running the workload. In order to fill the VMDK, we used the DiskTool program to fill random data over the entire VMDK.

**Figure 4** shows that SEsparse performs on par with or better than VMFSsparse for all the cases. Compared to random I/O performance on the local storage test bed, the performance gap between thin format and sparse formats is minimal in the SAN array. This is because even though sparse VMDK formats have to access more metadata information compared to the thin format, the overhead of a few extra seeks is absorbed by multiple disk drives in the SAN array. Whereas in the case of local storage test bed, all the I/Os and metadata accesses are served from a single disk drive, and therefore the overhead of extra seeks is reflected more in the performance results.
Hadoop MapReduce Applications

In addition to the Iometer-based performance tests, it is our intent to provide insight into how well different virtual disk technologies that VMware provides perform in real, I/O-intensive applications.

The need for processing and analyzing large amounts of unstructured data is today’s reality. Hadoop provides a scalable and flexible software framework solution for running distributed applications on clusters of commodity hardware [7]. Hadoop was architected to allow petabytes of data to be processed with a quick turnaround. The framework is built with two main components: data, represented by the distributed file system (HDFS) and compute, represented by the MapReduce computation engine.

During the life of a typical Hadoop job, the MapReduce tasks operate on data hosted in HDFS while saving intermediate results on temporary storage. In general, the MapReduce applications are I/O-intensive with a mixed access pattern, which makes them good candidates for evaluating the performance of virtual disk technologies offered by VMware. Moreover, traditionally HDFS and temporary data are hosted on shared physical drives. The amount of temporary space needed depends on the type of the workload; therefore, over-provisioning storage could easily lead to wasted space. Furthermore, temporary data is ephemeral and it will be cleared out at the end of the run. For example, the rule of thumb for the TeraSort workload is that it needs temporary storage—as much as twice the amount of the HDFS data it operates on—while the Pi workload doesn’t use any temporary storage. Because it can reclaim unused storage space, hosting both HDFS and temporary data on SEsparse delta disks is an excellent solution to help mitigate wasteful allocations. Moreover, shrinking on demand allows a better utilization of the existing physical storage and eliminates the need for additional storage devices.†

The potential of running Hadoop MapReduce applications in a virtualized environment is not the object of this paper. This subject is discussed in detail in “Virtualized Hadoop Performance with VMware vSphere 5.1,” and

† Space reclamation is an SEsparse feature not currently supported with Linux guest operating systems.
“Protecting Hadoop with VMware vSphere 5 Fault Tolerance” [8] [9]. The focus of this work is to show how well thin, VMFSsparse, and SEsparse virtual disk technologies are suited for hosting Hadoop data and its effect on MapReduce application runtime performance.

Cloudera's cdh3u4 Hadoop distribution offers a suite of benchmarks that can be used to test Hadoop's MapReduce performance on any given cluster configuration. Among these, we chose the ones considered most representative of a real Hadoop workload:

- TeraGen - creates the dataset to be sorted and its I/O pattern is mostly sequential writes of large blocks to HDFS.
- TeraSort - sorts the data generated by TeraGen. The application's I/O pattern is a mix of HDFS and temp reads and writes.
- TeraValidate - validates the correctness of the results produced by TeraSort. Its I/O pattern is mostly reads.

**Test-Bed Architecture**

![Hadoop cluster configuration](image)

*Figure 5. Hadoop cluster configuration*
## Test Environment

<table>
<thead>
<tr>
<th>Hardware</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server</strong></td>
<td>1 Dell PowerEdge C2100</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Intel Xeon Processor X5670 - dual socket, six - 2.93 GHz cores/socket</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>148GB RAM</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>LSI MegaRAID SAS 9260-8i, default controller cache configuration (512MB WB cache) containing 24 - 550GB, 15K RPM SAS internal drives</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BIOS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Hyper-Threading</td>
<td>Enabled</td>
</tr>
<tr>
<td>Power states</td>
<td>Disabled</td>
</tr>
<tr>
<td>TurboMode</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Host software configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VMware vSphere</strong></td>
<td>5.5 RC (build #1192841)</td>
</tr>
<tr>
<td><strong>VMFS</strong></td>
<td>Version 5.54</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hadoop cluster configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Six node Hadoop virtual cluster</strong></td>
<td>1 master VM with 4 vCPUs, 30GB RAM</td>
</tr>
<tr>
<td></td>
<td>5 worker VMs, each with 4vCPUs, 20GB RAM</td>
</tr>
<tr>
<td></td>
<td>Ubuntu 12.04 LTS</td>
</tr>
<tr>
<td></td>
<td>Guest file system: ext4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Each Hadoop node VM was equipped with</th>
<th>1 PVSCSI adapter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 virtual disks</td>
</tr>
<tr>
<td></td>
<td>2 virtual NICs (VMXNET3)</td>
</tr>
</tbody>
</table>

| Hadoop distribution | Cloudera cdh3u4 (hdfs2, mr1) |

<table>
<thead>
<tr>
<th>Hadoop configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HDFS block size</strong></td>
<td>256MB</td>
</tr>
<tr>
<td><strong>Concurrent mappers and reducers per cluster node</strong></td>
<td>8</td>
</tr>
<tr>
<td><strong>JVM parameters Map task</strong></td>
<td>-Xms800m -Xmx800m -Xmn256m</td>
</tr>
<tr>
<td><strong>JVM parameters Reduce task</strong></td>
<td>Xms1200m -Xmx1200m -Xmn256m</td>
</tr>
</tbody>
</table>
Test Methodology

All six Hadoop node VMs ran on a single ESXi host. While the master VM ran Hadoop NameNode, JobTracker, DataNode, and TaskTracker; the five worker VMs each ran a single instance of DataNode and TaskTracker. The VMs were equipped with three VMDKs shared between HDFS and temp data. Each VMDK was stored on a single VMFS data store spanning an entire physical drive. There was no sharing of the same data store between two or more virtual disks. An ext4 file system was created inside the VM on a 4KB aligned partition. The sparse virtual disks used in this performance study were created from a thin VMDK parent using the default block size for VMFSsparse and 4KB blocks for SEsparse.

Each VM was configured with two virtual NICs. Hadoop network traffic was routed over a private subnet and a vSwitch exclusively dedicated to it. Since the entire Hadoop cluster was running on a single host, we were able to leverage the high throughput and low latency provided by the vSwitch technology.

A single test consisted of a TeraGen, TeraSort, and TeraValidate sequence running on a 500GB data set. Each experiment was repeated three times to account for potential variation in the results. Before every test sequence a fresh set of virtual disks was created and attached to the Hadoop node VMs. Both VMFSsparse and SEsparse delta disks were created using a thin base virtual disk. Given the size of the TeraSort input data set used in the experiments, and the total storage provisioned in our Hadoop’s cluster deployment, write to empty disk performance will be more relevant for TeraGen. TeraSort will write data to non-allocated regions of the disks while reading allocated data; and finally, TeraValidate will read allocated data while writing data to non-allocated regions of the disks.

Results

The graph below showcases the performance of TeraGen, TeraSort, and TeraValidate expressed in elapsed time for each benchmark to run to completion. The data points in the graph represent averages over the three executions of the test applications².

VMFSsparse shows better performance than thin VMDKs and SEsparse on the TeraGen workload, which consists mostly of sequential writes. The sequential write performance on the thin format is hurt by the fact that with an empty thin VMDK the block is only written after it is zeroed out on disk. SEsparse shows approximately 40% lower sequential write performance than VMFSsparse due to extra metadata housekeeping operations executed on every write to SEsparse.

TeraSort performs best on thin, while VMFSsparse performance is better than SEsparse by approximately 5%. This is somewhat expected given the performance differences between VMFSsparse and SEsparse observed for TeraGen and TeraValidate and the I/O pattern in TeraSort, which is 60% writes and 40% reads.

TeraValidate, which exclusively does read operations, performs best on thin VMDKs while SEsparse performance is better than VMFSsparse by 43%.

In summary, for many real-world MapReduce applications that are a mixture of I/O and CPU operations, SEsparse performance is close to VMFSsparse.

² Although the capabilities of a single internal drive estimated using microbenchmarks could be an indicator of the overall performance of a MapReduce workload running on Hadoop, it is not practical to extrapolate the performance of a real-life application running on a large-scale Hadoop cluster considering only a single drive, due to additional factors coming into play which can affect performance like VM scheduling, LSI controller scalability, and saturation of hardware resources.
VDI Workload (View Planner)

VMware View Planner is used to generate a VDI workload on virtual desktops while measuring operation latencies on the client side. This end-to-end latency is representative of what the user sees while using a VDI desktop [2]. The latency measured at each client is aggregated into two categories: CPU-sensitive (Group A) and storage-sensitive (Group B) measures, for the whole run. The standard workload consists of productivity applications (Microsoft Word, Excel, and PowerPoint), web browser applications (Internet Explorer, Firefox, and Web Album Browse), compression using 7zip, 720p video watching, and PDF browsing. In order to better simulate the creation and deletion of file system blocks, we used the “Custom Apps” feature of View Planner to add a workload that installs and uninstalls a software application. This creates reclaimable space to measure SEsparse wipe-shrink performance. Such reclaimable space could be created by any operation that deletes files on the desktop such as deleting downloaded content, uninstalling applications, and running disk cleanup utilities.

Test Scenario and Environment

The View Planner workload was run on linked clones using the VMFSsparse and SEsparse disk formats (thin disk format is not supported for linked clones). To understand the performance impact of storage reclamation in the SEsparse case, three sub-scenarios were tested: before running wipe-shrink, during wipe-shrink, and after running wipe-shrink.

Figure 7 shows the generic VMware View Planner architecture. In this particular test, three hosts were used, one each for desktop VMs, client VMs, and other infrastructure VMs (like VMware vCenter, VMware Horizon View⁹, Active Directory, databases, and the View Planner appliance). The View Planner appliance is the test-driver, which powers on the client and desktop VMs, initiates one-to-one PCoIP remote sessions from client to desktop VMs, and starts the View Planner workload in all the desktop VMs. The client VMs monitor the workload progress and measure end-to-end latency.

---

⁹ An internal build of Horizon View that supports vSphere 5.5 was used.
Figure 7. VMware View Planner architecture

<table>
<thead>
<tr>
<th><strong>Host Details</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host</strong></td>
<td>HP ProLiant BL460cG6, 2xQuad-Core IntelXeon <a href="mailto:E5540@2.53GHz">E5540@2.53GHz</a>, HT enabled, 96GB RAM</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>EMC VNX 5700 Non-SSD - 3TB RAID0 LUN with 6 SAS2 HDD connected using 8G FC</td>
</tr>
<tr>
<td><strong>Desktop VM Details</strong></td>
<td>Windows7 SP1 32bit, 1 vCPU, 1GB RAM, 16GB VMDK, LSI Logic SAS, VMXNET3</td>
</tr>
<tr>
<td><strong># of Desktop VMs</strong></td>
<td>40 Linked Clones</td>
</tr>
<tr>
<td><strong>View Planner Details</strong></td>
<td>Remote Mode, 5 Iterations, 10 seconds think time</td>
</tr>
<tr>
<td><strong>Workload</strong></td>
<td>All Standard Apps + InstallUninstallApp</td>
</tr>
<tr>
<td><strong>VDI Software Stack</strong></td>
<td>VMware Horizon View 5.2, VMware vSphere 5.5 Build#1192841</td>
</tr>
</tbody>
</table>

Table 4. Test-bed configuration details

Results

View Planner ran the standard five iterations. For each iteration, we ran all applications and measured end-to-end latency at the client. We considered only the middle three iterations as steady state and used these in the figures below because of the ramp-up and ramp-down effects of boot storm and login storm. We ran the tests three times and averaged the results to ensure reproducibility and measure confidence intervals. The wipe-shrink operation was done in batches of 8 VMs to equally distribute the load throughout the run.

Even though Figure 8 shows a slight increase in the latencies of CPU sensitive View Planner operations for SEsparse (before wipe-shrink) relative to the redo-log, these numbers are within the margin of error for the runs and are not statistically significant. The slight improvement in SEsparse performance after wipe-shrink is expected due to less fragmented VM disks.
Similarly, Figure 9 shows the performance impact on storage-sensitive operations with some increase in latency during wipe-shrink and improvements after wipe-shrink.

Figure 10 shows a 10% CPU cost for running wipe-shrink operations in SEsparse (based on our batch size). Even though it does not lead to any perceivable difference in user latencies, we recommend you schedule this operation during low server load.

Space reclamation of around 600MB per VM was seen during the run. This depends on the size of the application that was uninstalled (130MB per iteration in this case). In production environments, we recommend you configure the Windows Disk Cleanup utility to automatically run in the maintenance window. If Disk Cleanup is used to delete temporary internet files, service pack backup files, recycle bin, and other temporary/dump files, we expect users to see more than 1GB of space reclamation per VM over several weeks.
Conclusion

This paper presents a performance study comparing sparse virtual disk formats in VMFS, namely VMFSsparse and SEsparse, using the thin virtual disk format as the baseline. The performance results are from the Iometer micro-benchmark (on two setups with different classes of storage), and two real world application domains: Big Data Analytics and VDI. Overall, we show that the SEsparse virtual disk format performs mostly on par with, or better than the VMFSsparse format (depending on the workload), while also having the ability to reclaim storage space freed up by the guest.
References


About the Authors

Dr. Razvan Cheveresan is a senior performance engineer in the VMware Performance Engineering group. He is currently working on various performance aspects of big data virtualization. He has a PhD in Computer Science from “Politehnica” University of Timisoara.

Dr. Sankaran Sivathanu is a senior engineer in the VMware Performance Engineering group. His work focuses on the performance aspects of the ESXi storage stack and characterization/modeling of new and emerging I/O workloads. He has a PhD in Computer Science from the Georgia Institute of Technology.

Tariq Magdon-Ismail is a staff engineer at VMware. He has over 15 years of industry experience working on large-scale systems, performance, and scalability. Tariq’s current research interests sit at the intersection of systems virtualization and Big Data performance.

Aravind Bappanadu is a senior engineer in the VMware Performance Engineering group. He works on performance efforts in VMware’s VDI and disaster recovery products.

Acknowledgements

The authors thank their colleagues in the Performance team and in other VMware departments for their careful reviews and valuable feedback.