Understanding Clones in VMware vSphere 7

Performance Study - May 27, 2021

Revised to include performance testing with VMware Virtual Volumes (vVOLs) and VMware vSAN

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com

Table of Contents

1 Executive Summary

Many organizations have successfully adopted VMware vSphere® clones to help meet customer demands faster and to achieve business agility. However, development and operations teams are often faced with questions on the performance aspects and deployment rates of the different types of clones offered on the vSphere platform.

vSphere offers three different types of clones: full clone, linked clone, and instant clone. In this paper, we evaluate clone performance with a variety of workloads and discuss the provisioning rates of the different clone types. In addition, we provide some tips and tricks to improve the provisioning rates.

2 Introduction

Installing a guest operating system and applications can be time consuming. With clones, you can conveniently make copies of a virtual machine. vSphere offers three different types of clones.

Full Clone

A full clone is an independent child VM that shares nothing with the parent VM after the cloning operation. The ongoing operation of a full clone is entirely separate from the parent VM.

Linked Clone

A linked clone is a child VM that shares virtual disks with the parent VM in an ongoing manner. A linked clone is made from a snapshot of the parent and uses snapshot-based delta disks. The child disks employ a copy-on-write (COW) mechanism, in which the virtual disk contains no data in places until copied there by a write. This optimization conserves storage space.

Instant Clone

An instant clone is a child VM that shares virtual disks and memory with the parent VM in an ongoing manner. Like the linked clone, an instant clone also leverages delta disks to conserve storage space. An instant clone also shares the memory state of the parent VM to deliver efficient memory use. Instant clones are always created from a running VM instead of a powered-off VM, and function much like a container. Instant clones, unlike full clones or linked clones, are always created in a powered-on state with the VM ready for users to connect with the guest applications running inside the parent VM prior to provisioning.

Because a full clone does not share virtual disks or memory with the parent virtual machine, they usually perform better than linked clones or instant clones. However, full clones take longer to provision than the other clone types.

3 Scope of Paper

The behavior of the clones provisioned either from the vSphere Client, pyVmomi, PowerCLI, or any other SDK client will be the same, since they all rely on a common vSphere API, and this exposes the functionality to provision a clone or customize a guest. The choice of SDK depends on your preference.

We will focus on pyVmomi-based programs, although the same principles apply to other languages like VMware PowerCLI™. (*pyVmomi* is the Python SDK for the VMware vSphere API that allows you to manage vSphere and VMware vCenter Server®.) We focus only on the performance aspects and provisioning rates of different clone types. Considerations such as guest customization for instant clones are outside the scope of this paper.

Additional resources:

- vSphere Web Services SDK
  https://code.vmware.com/docs/11721/vmware-vsphere-web-services-sdk-programming-guide/
- VMware APIs and SDKs
  https://code.vmware.com/sdks
- Guest customization support for instant clones
  https://www.virtuallyghetto.com/2020/05/guest-customization-support-for-instant-clone-in-vsphere-7.html

4 Performance Test Configuration and Methodology

This section describes the testbed configuration, workloads, and general testing methodology.

4.1 Testbed configuration

The testbed consisted of two Dell R930 servers running VMware ESXi™ 7.0 U1. Each quad-socket server had two Intel Xeon E7-8890 processors with 24 cores, ran at 2.20 GHz, and had 4TB memory. Each host had two Intel 10GbE network adapters. The management and provisioning networks used a unique 10GbE network adapter. Storage included a 10TB VMFS volume on a Dell EMC Unity 600 all-flash array, and a 2TB VMFS volume on a Dell EMC VNX7600 all-flash array.

4.2 Workloads

In this study, we used several workloads, which we describe next.

4.2.1 OLTP database workload

We used the open-source HammerDB (https://hammerdb.com/) as the client load generator with the TPC-C (http://www.tpc.org/tpcc/) workload profile. The performance metric is transactions per minute (TPM).

> NOTE: This is a non-compliant, fair-use implementation of the TPC-C workload for testing the benefits of a feature; our results should not be compared with official results.

Software:

- VM config: 12 vCPUs, 32GB memory, 3 vmdk files (100GB system disk, 250GB database disk, 100GB log disk)
- Guest OS/Application: RHEL 7.6 / Oracle 12.2
- Benchmark: HammerDB using a TPC-C database size of 1000 warehouses
- HammerDB client: 10 users with 500ms think-time, 5 min ramp-up, 5 min steady-state

4.2.2 JVM workload

We used the industry standard SPECjbb 2015 (https://www.spec.org/jbb2015/) as the client load generator. The performance metric is average processed requests (PR) over 10 iterations.

> NOTE: Our results are not comparable to SPEC's trademarked metrics for this benchmark.

Software:

- VM config: 4 vCPUs, 32GB memory, 1 vmdk file (16 GB disk)
- Guest OS/Application: CentOS / JVM
- SPECjbb benchmark parameters:

```
JAVA_OPTS : -Xms30g -Xmx30g -Xmn27g -XX:+UseLargePages
-XX:LargePageSizeInBytes=2m -XX:-UseBiasedLocking -XX:+UseParallelOldGC

specjbb.control (type/ir/duration): PRESET:10000:300000
```

4.2.3 In-memory, key-value store workload

We chose the open-source Redis (https://redis.io/) as the guest application and the open-source memtier_benchmark (https://redislabs.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis-memcached/) as the client load generator. Redis is a popular in-memory, key-value store that can be used as a database, cache, or message broker. The performance metric is the average time to run 40 million operations.

Software:

- VM config: 4 vCPUs, 12GB memory, 2 vmdk files (100GB OS disk, 100GB Redis/apps disk)
- Guest OS/Application: RHEL 7.6 / Redis
- Memtier benchmark parameters (2 sets of performance data with different SET:GET ratio):

```
memtier_benchmark -t 10 -n 200000 --ratio 100:0 -c 20 --hide-histogram  -x 1 --key-pattern
R:R --distinct-client-seed -d 256 --pipeline=5

memtier_benchmark -t 10 -n 200000 --ratio 0:100 -c 20 --hide-histogram  -x 1 --key-pattern
R:R --distinct-client-seed -d 256 --pipeline=5
```

4.2.4 Virtual desktop workload

VMware Horizon (https://www.vmware.com/products/horizon.html) technology provides a secure delivery of virtual desktops and apps across hybrid and multi-cloud deployments. We chose VMware View Planner (https://www.vmware.com/products/view-planner.html) as the workload to measure the performance of a virtual desktop deployment platform.

Software:

- VM config: 2 vCPUs, 4GB memory, 100GB thin disk (20GB consumed storage)
- Guest OS/Application: Windows 10 / View Planner
- View Planner run mode: Local

4.2.5 FIO workload

FIO is an I/O microbenchmark used to measure file system I/O performance. The performance metric is I/Os per seconds (IOPS).

Software:

- VM config: 4 vCPUs, 32GB memory, 2 vmdks (100GB system disk, 50GB data disk)
- FIO benchmark parameters (2 sets of performance data with random I/O and sequential I/O):

```
-ioengine=libaio -iodepth=32 –rw=randrw –bs=4096 -direct=1 -numjobs=4 -group_reporting=1 -
size=50G –time_based -runtime=300 -randrepeat=1

-ioengine=libaio -iodepth=32 –rw=readwrite –bs=4096 -direct=1 -numjobs=4 -group_reporting=1 -
size=50G –time_based -runtime=300 -randrepeat=1
```

5 Single clone provisioning time and performance

We first discuss the creation of a single clone, followed by a comparison of performance and provisioning time.

5.1 Creating a clone

You can create a full clone using the vSphere Client, which provides the workflow to create a new full clone with a new MAC address and a unique identifier for the clone. You must create linked and instant clones through the ESXi host's command-line interface.

We used the following pseudo code.

5.1.1 Creating a Full Clone

```
fcspec = vim.vm.Clonespec()
relocateSpec = vim.vm.RelocateSpec()
relocateSpec.diskMoveType =
vim.vm.RelocateSpec.DiskMoveOptions.moveAllDiskBackingsAndDisallowSharing
fcspec.location = relocateSpec

vm.Clone(vm.parent, fcname, fcspec)
```

5.1.2 Creating a Linked Clone

```
lcspec = vim.vm.Clonespec()
relocateSpec = vim.vm.RelocateSpec()
relocateSpec.diskMoveType = vim.vm.RelocateSpec.DiskMoveOptions.createNewChildDiskBacking
lcspec.location = relocateSpec

vm.Clone(vm.parent, lcname, lcspec)
```

5.1.3 Creating an Instant Clone

```
icspec = vim.vm.InstantCloneSpec()
icspec.location = vim.vm.RelocateSpec()
icspec.name = icname

vm.InstantClone(icspec)
```

Please refer to section "11 Appendix" for the full code example to create an instant clone.

5.2 Test workflow

We used the following general workflow to compare the performance of different types of clones.

Clone	Test Workflow
Full Clone	• Create a full clone (child VM) from the baseline VM • Measure the performance of the full clone
Linked Clone	• Create a new parent VM from the baseline VM • Power on the parent VM • Create a snapshot of the parent VM • Create a linked clone (child VM) from the parent VM • Measure the performance of the linked clone
Instant Clone	• Create a new parent VM from the baseline VM • Power on the parent VM • Bring up all the guest applications • Create an instant clone (child VM) from the parent VM • Measure the performance of the instant clone

Table 1: Test workflow

5.3 Performance and provisioning time

In this section, we focus on the performance and provisioning time of various types of clones created on the VMFS datastore. In a later section, we explore the performance aspects of clones on other datastores, including vSAN and vVOL.

5.3.1 HammerDB workload

Figure 1 below compares the performance and provisioning time of the three types of clones when we ran the HammerDB workload on the VMFS datastore. The test scenario used a local clone operation, which means we created the child VM on the same host as the parent VM.

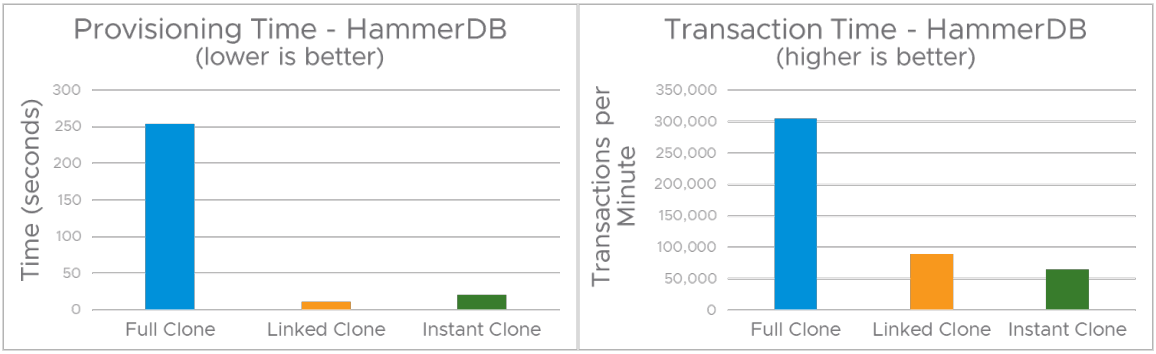

Figure 1: Provisioning time and performance of a single clone VM while running the HammerDB workload on the VMFS datastore

As shown in figure 1 above, the provisioning time of the linked clone is 23x faster than the full clone, and the instant clone is 12x faster because copying the entire disk contents during full clone creation is time consuming. There is a slightly higher provisioning time of the instant clone (21 seconds) compared to the linked clone (11 seconds) because of the requirement to mark all the memory pages of the parent as copy-on-write. Note this is a one-time operation, and hence the provisioning time of subsequent instant clones from the same parent will be more like that of the linked clone.

Figure 1 also shows the performance of the full clone is much greater than that of the linked clone and instant clone for disk I/O–intensive workloads. This is because vSphere manages the snapshot-based delta disks on a VMFS datastore using traditional redo logs, which results in additional disk I/O latency, thereby significantly impacting the guest performance for I/O-intensive workloads. This latency is expected since linked and instant clones use snapshot-based delta disks that contain no data until copied there by a write. The read requests to a region that has been written to delta disks is returned from the delta disks, and the read requests to a region not written to delta disks are redirected to the parent VM's disks. This optimization conserves storage space, but it has an impact on performance, especially for disk I/O–intensive workloads. While the HammerDB workload is quite disk I/O–intensive, it also puts stress on other system resources including CPU and memory. Because of this, the performance of the instant clone is a little lower than the linked clone, because the linked clone does not share any memory state with the parent VM.

5.3.2 SPECjbb workload

Figure 2 below compares the provisioning time and performance of the three types of clones when running the SPECjbb 2015 workload.

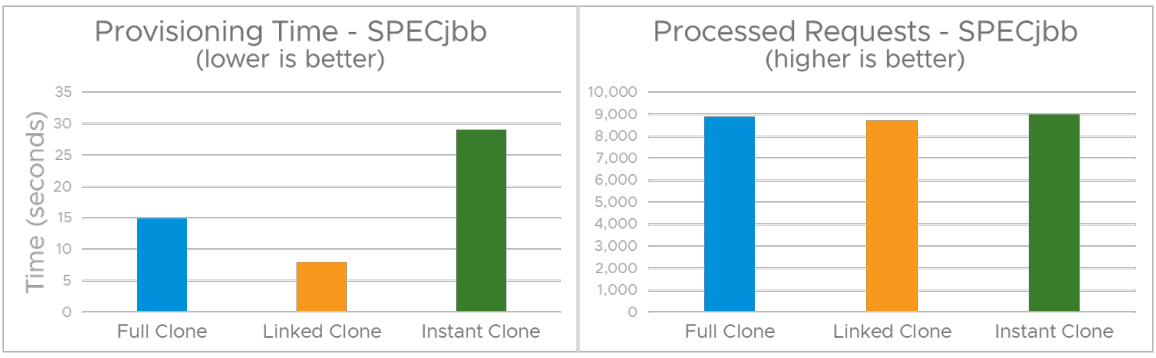

Figure 2: Provisioning time and performance of a single clone VM running the SPECjbb 2015 workload

First, let us focus on the performance. In contrast to the performance differences observed among the clones with the HammerDB workload, figure 2 shows nearly identical performance for all three clones when running the SPECjbb workload. The difference in the performance metric (SPECjbb processed requests) among the three types of clones is in the noise range (< 5%). This shows that while linked and instant clones are not good candidates for disk I/O–intensive workloads, they perform well for CPU- and memory-intensive workloads.

The provisioning time of the full clone shown in figure 2 was only 15 seconds compared to the full clone provisioning time of 254 seconds during the previous test scenario with the HammerDB workload (shown in figure 1). This time difference occurs because the VM was configured with a single 16GB vmdk compared to the HammerDB scenario where the VM was configured with 3 vmdk files, totaling 450GB storage. This confirms that full clone provisioning time is proportional to the VM storage. Once again, we see the provisioning time of the instant clone was higher compared to the linked clone due to the requirement to mark all the memory pages of the parent as copy-on-write.

5.3.3 Memtier benchmark

Config	Provision Time	Execution Time (100% SET)	Execution Time (100% GET)
Full Clone	95 seconds	144 seconds	125 seconds
Linked Clone	9 seconds	144 seconds	125 seconds
Instant Clone	7 seconds	148 seconds	125 seconds

Table 2: Provisioning time and performance of a single clone VM running the SPECjbb 2015 workload

Table 2 compares the provisioning time and performance of the clones when running the Redis/Memtier workload.

Similar to the SPECjbb results, we observe nearly identical performance among all of the clones in the two test scenarios in which we varied the GET:SET mix. This is expected because the Redis/Memtier workload is CPU- and memory-intensive with no disk I/O component. Once again, we see the provisioning time of the linked clone and the instant clone to be a magnitude lower than the full clone provisioning time.

5.3.4 Performance impact on parent VM

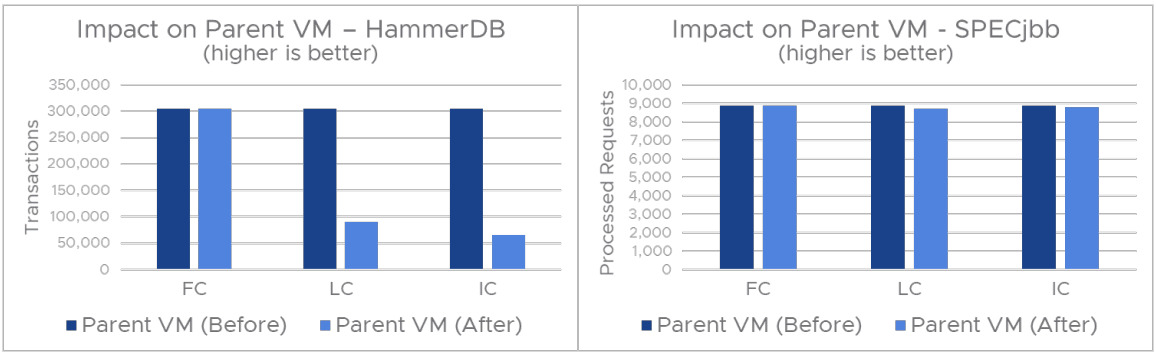

Figure 3: Performance impact on parent VM after provisioning operation on VMFS datastore

Figure 3 shows the performance impact of running a parent VM before and after the creation of a clone. Creating a full clone has no impact on the parent VM performance for both HammerDB and SPECjbb workloads. This is expected because a full clone is an independent child VM that shares nothing with the parent VM after the cloning operation. In comparison, when an instant clone or a linked clone is provisioned, the parent VM also gets new snapshot-based delta disks because both clones share virtual disks with the parent VM in an ongoing manner. In the case of the instant clone, the parent VM also shares the memory state with the child VM. The performance of the parent VM after a provisioning operation will be like that of the provisioned child VM.

In summary, we draw the following observations from the three workloads in a single-clone test scenario:

- The provisioning time of a full clone is bound by two factors:
  - The total VM storage size
  - The disk I/O throughput of source and destination datastores

> NOTE: The VM storage size refers to the actual storage space used by the VM. The unused storage space of a VM's thin disks has no impact on the provision time.

- Provisioning time of the instant and linked clones does not vary based on the VM size or disk I/O throughput.
- Linked and instant clones are not good candidates for disk I/O–intensive workloads.
- Each of the clones perform well with CPU- and memory-intensive workloads.
- Performance of the parent VM after a provisioning operation is similar to that of the child VM provisioned.

6 Remote Clone Provisioning Time

An instant clone can be created only on the host of the parent VM. A linked clone can be created on a different host from the parent VM in the presence of shared storage. In comparison, a full clone can be created on a remote host without any dependency.

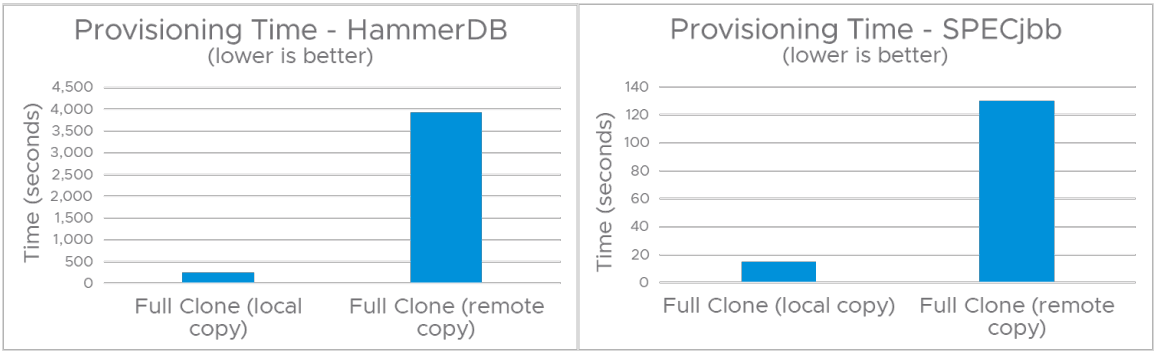

Figure 4: Local and remote provisioning time of full clones

Figure 4 shows the performance implications of cloning over a network. A full clone operation prefers to use the host's storage interface (the host that has access to both source and destination datastores) to transfer the disk content, thus reducing the provisioning network utilization and the host's CPU utilization. If neither the host of the parent VM nor the host of the child clone have access to both source and destination datastores, cloning operations use the network file copy (NFC) service to copy the parent's VM virtual disks over a network.

We observed that NFC (which uses nfclib) is inefficient compared to the host's storage interface (which leverages disklib/DataMover). We found the provision time of a remote full clone to be 15x slower than that of a local full clone for the HammerDB VM, and 9x slower for the SPECjbb VM. Our performance testing also showed peak network bandwidth used during the remote clone operation ranged between 500Mbps to 1.2Gbps based on type of the vmdk files (thick or thin) and disk content. Moreover, the NFC service does not leverage multiple vmknics. We are looking to add new optimizations to ESXi to reduce the provisioning time of the remote clone operation.

7 Multiple Clone Performance and Provisioning Time

In this section, we compare the performance and provisioning times of different types of clones (instant, linked, and full clones) in the presence of multiple clones with HammerDB and SPECjbb workloads on a VMFS datastore.

7.1 Performance

Figure 5 shows the aggregate performance of child VMs running the HammerDB workload.

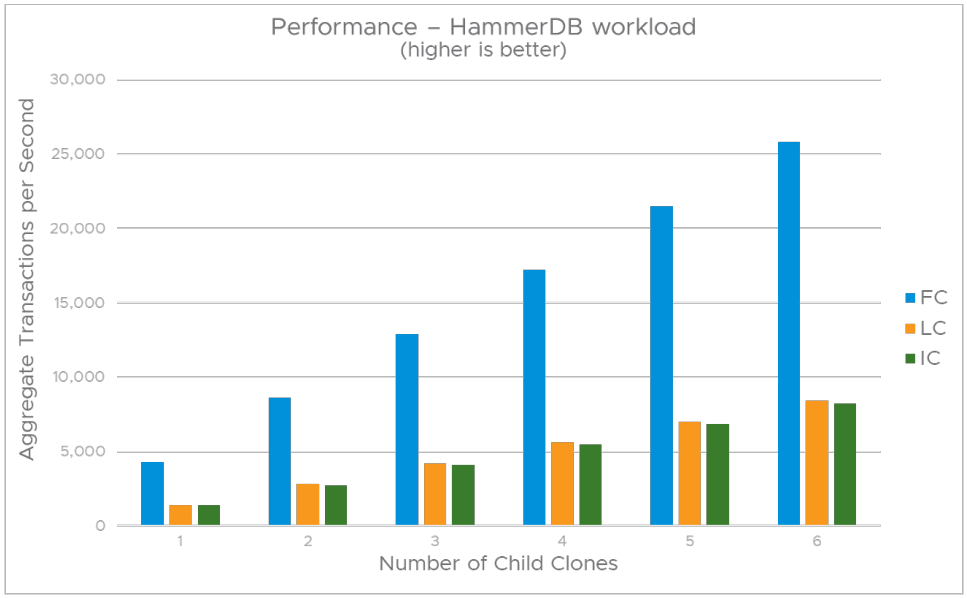

Figure 5: Aggregate performance of multiple clone VMs running the HammerDB workload on the VMFS datastore

In our tests, we varied the number of child VMs from 1 to 6. We observed that the performance (aggregate transactions per second) of linked and instant clones is very similar, and 1/3rd of the performance of full clones. This poor performance of linked and instant clones is consistent with the results of the single clone/HammerDB test scenario, and it is due to the presence of redo logs resulting in additional disk I/O latency.

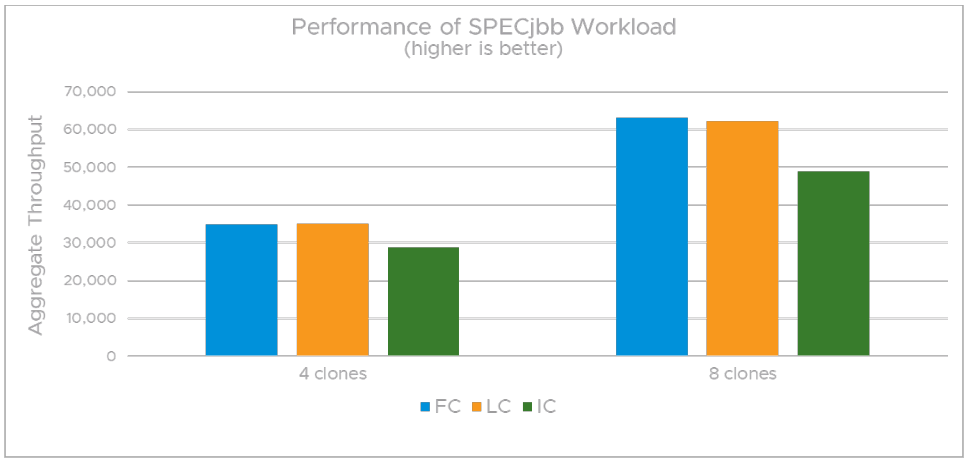

Figure 6: Aggregate performance of multiple clone VMs running the SPECjbb workload

Figure 6 shows the aggregate performance of child VMs when running the SPECjbb 2105 workload. In our tests, we varied the number of child VMs from 4 to 8. First, we observe that performance (aggregate transactions per second) of full clones and linked clones is nearly identical in both 4-clone and 8-clone scenarios, as seen in the single clone performance scenario earlier in this paper. This is not surprising since SPECjbb 2015 is very CPU- and memory-intensive, but it has no disk I/O component. For workloads with a minimal disk-I/O component, we expect the performance of linked and full clones to be nearly identical.

Figure 6 also shows the performance of instant clones is about 20 percent lower than that of linked and full clones, in the presence of multiple instances of child VMs. This performance impact with instant clones is expected when running memory-intensive workloads, since all the instant clones share the same physical memory pages as their parent VM.

7.2 Provisioning time

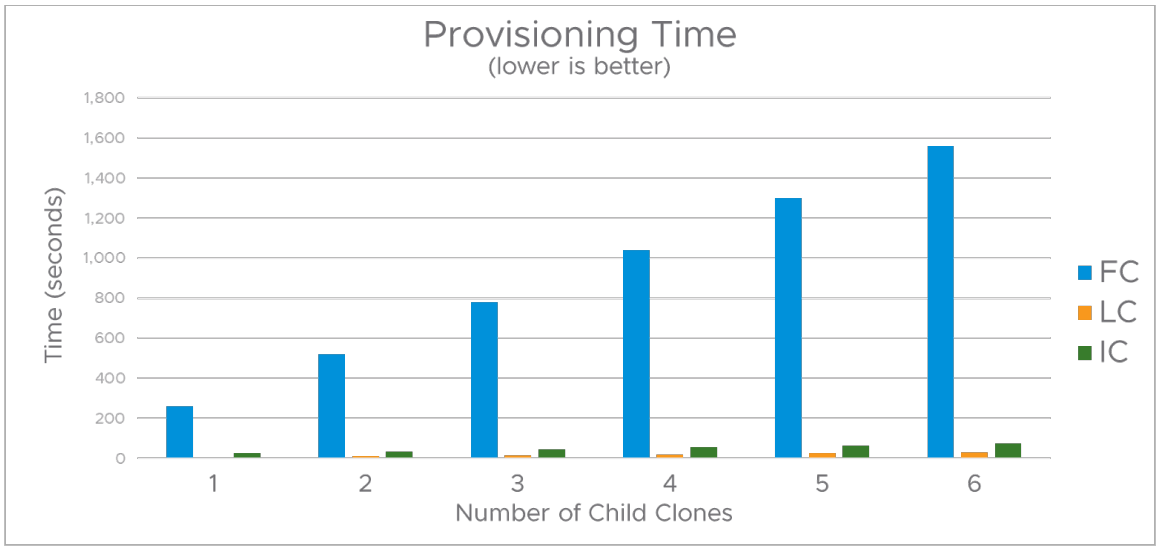

Figure 7: Provisioning time of multiple clone VMs running the HammerDB workload

Figure 7 compares how the provisioning times of different types of clones increases with an increase in the number of child clones. Note that the clones were created in a serial fashion, one clone at a time. In a later section, we explore provisioning multiple clones in parallel.

When creating clones serially, we observe the total full clone provisioning time increases linearly with the number of child clones provisioned. In general, we note that the provisioning time of linked and instant clones is faster than full clone provisioning time.

Instant clones have a slightly higher provisioning time than linked clones, but instant clones are always created in a powered-on state, ready for users to connect to with guest applications running inside the parent VM prior to the provisioning.

7.3 Understanding instant clone provisioning time

Figure 8 shows the impact of VM memory on instant clone provisioning time.

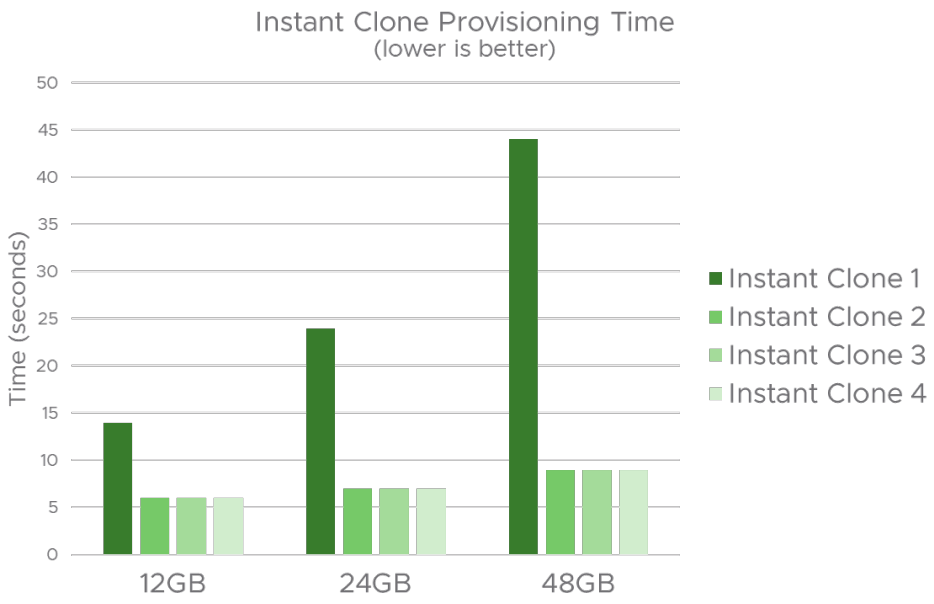

Figure 8: Provisioning time of instant clones

Note that VM memory here does not refer to the VM assigned memory, instead it refers to the physical memory granted to the VM by ESXi. In other words, it refers to the memory that has been touched by the guest applications running inside the parent VM.

We considered three scenarios in which we varied the VM memory size from 12GB to 48GB. In each of the scenarios, we created four instant clones from the same parent. From these tests, we observe two things. First, the provisioning time of the first instant clone is proportional to the VM memory because of the requirement to mark all the memory pages of the parent VM as copy-on-write. We also observe this cost is a one-time operation that is incurred during the first instant clone creation, hence the provisioning time of subsequent instant clones from the same parent is much shorter. We are looking to add new optimizations to ESXi to reduce the provisioning time of the first instant clone.

8 Bulk Provisioning

It is common for cloud services to deploy tens or hundreds of instances of a single template VM across a datacenter with prolonged deployment time. In this section, we discuss the provisioning rate and make suggestions to improve it.

8.1 Creating the clones

We used the following pseudocode to create clones in a bulk fashion.

1. Bulk creating Full Clones pseudocode

```
namePrefix = fcname
locSpec = vim.vm.RelocateSpec()
fcspec = vim.vm.Clonespec(location=locspec, powerOn=False, template=False)

bulkFC = [(myVm.Clone,
       (vmFolder, namePrefix+str(vmIdx), fcspec))) for vmIdx in vmRange]
```

2. Bulk creating Linked Clones pseudocode

```
namePrefix = lcname
snapshot = myVm.snapshot.GetCurrentSnapshot()
locSpec = vim.vm.RelocateSpec()
locSpec.diskMoveType = vim.vm.RelocateSpec.DiskMoveOptions.createNewChildDiskBacking
lcspec = vim.vm.Clonespec(location=locSpec, powerOn=False, template=False,
snapshot=snapshot)

bulkLC = [(myVm.Clone,
       (vmFolder, namePrefix+str(vmIdx), lcspec))) for vmIdx in vmRange]
```

3. Bulk creating Instant Clones pseudocode

```
namePrefix = icname
locspec = vim.vm.RelocateSpec()

bulkIC = [(myVm.InstantClone,
       (vim.vm.InstantCloneSpec(name=namePrefix+str(vmIdx), location=locspec),)) for
vmIdx in vmRange]
```

8.2 Provisioning time

Figure 9 compares the provisioning time of one hundred SPECjbb local clones.

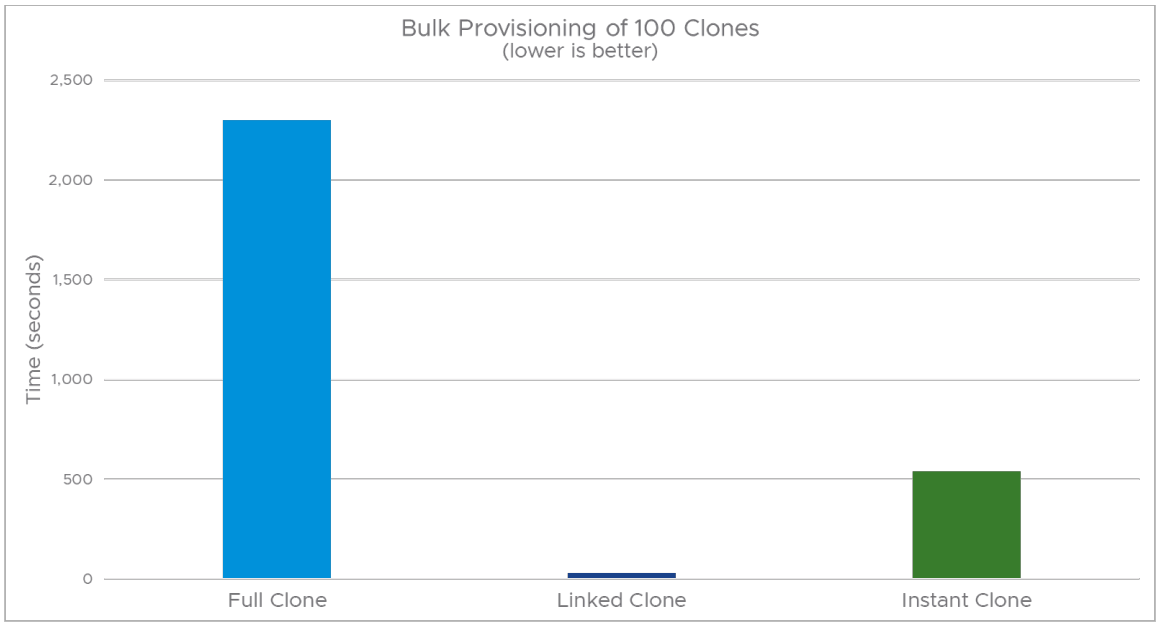

Figure 9: Bulk provisioning time of 100 SPECjbb clones

With only 30 seconds of provisioning time, the linked clone at 30 seconds performs much better than the full clone at 2,300 seconds and the instant clone at 539 seconds.

8.3 Full clone provisioning time: hot clone vs cold clone

By default, vCenter limits the concurrency of active clone operations to 1. There is an advanced vCenter setting `config.vpxd.ResourceManager.maxCostPerHost` that lets you increase the concurrency. However, it is easier to increase the concurrency by powering off the parent VM prior to bulk provisioning. This is because while vCenter limits the concurrency to 1 when cloning a running parent VM (hot-clone operation), the concurrency is increased to 8 when cloning a cold parent VM in a powered-off state (cold-clone operation).

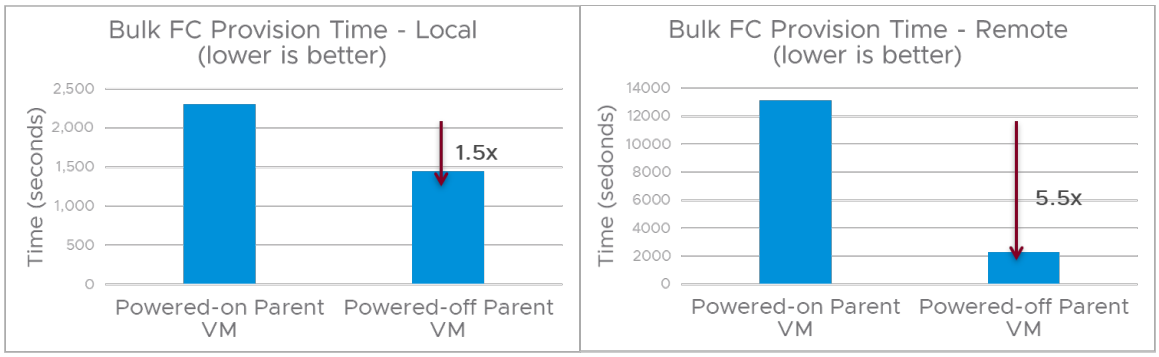

Figure 10: Bulk provisioning time of 100 full clones (FC)

In both the local clone and remote clone provisioning scenarios, we see a huge reduction in the provisioning duration time when the parent is powered off due to the higher concurrency of 8.

8.4 Reducing instant clone provisioning time

An instant clone can be created either from a running parent VM or a frozen parent VM. Figure 11 shows the differences between the two instant clone creation workflows.

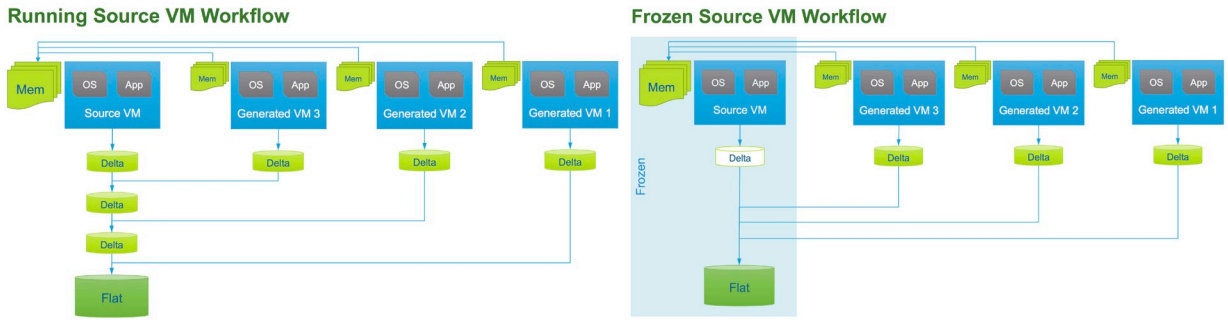

Figure 11: Instant clone creation workflows

In the first workflow, when an instant clone is created from a running parent VM, ESXi will briefly prevent the parent VM from running, take a checkpoint of the VM state, and mark all of its pages as copy-on-write. A new snapshot (delta disk) is created for each disk, enabling disk sharing with the new child VM.

In the second workflow, an instant clone is created from a *frozen* parent VM. A VM is frozen when a freeze operation is initiated from within the guest operating system using the VMware Tools vmware-rpctool utility and specifying the `instantclone.freeze` command. The source VM is no longer running after the freeze, and the disks of the parent VM are placed in read-only mode. Hence, the new delta disks are created only for the child VM. The parent VM continues to be in a frozen state until it is powered off. For more information on these workflows, please refer to "New

Instant Clone Architecture in vSphere 6.7" (https://www.virtuallyghetto.com/2018/04/new-instant-clone-architecture-in-vsphere-6-7-part-1.html). Below is an example used to freeze a running Linux VM:

```
// run the following command from a command terminal inside the guest
# vmware-rpctool "instantclone.freeze"
```

The end result of both workflows yields instant clones that behave the same. However, when deploying many instant clones, the second workflow is recommended both from the deployment standpoint and provision duration. With the first workflow (figure 11), since a new delta disk is created for each new instant clone, the parent VM's disk-chain depth can get deep, potentially hitting the disk-chain depth limits. While the provisioning time of the first instant clone is nearly the same for both workflows, the first workflow incurs additional latencies compared to the second workflow during each subsequent instant clone provisioning operation. For instance, the second workflow does not incur the latency of creating additional parent delta disks for each subsequent instant clone. Besides, there is no need to mark the parent pages as copy-on-write for each subsequent instant clone because the parent VM is frozen.

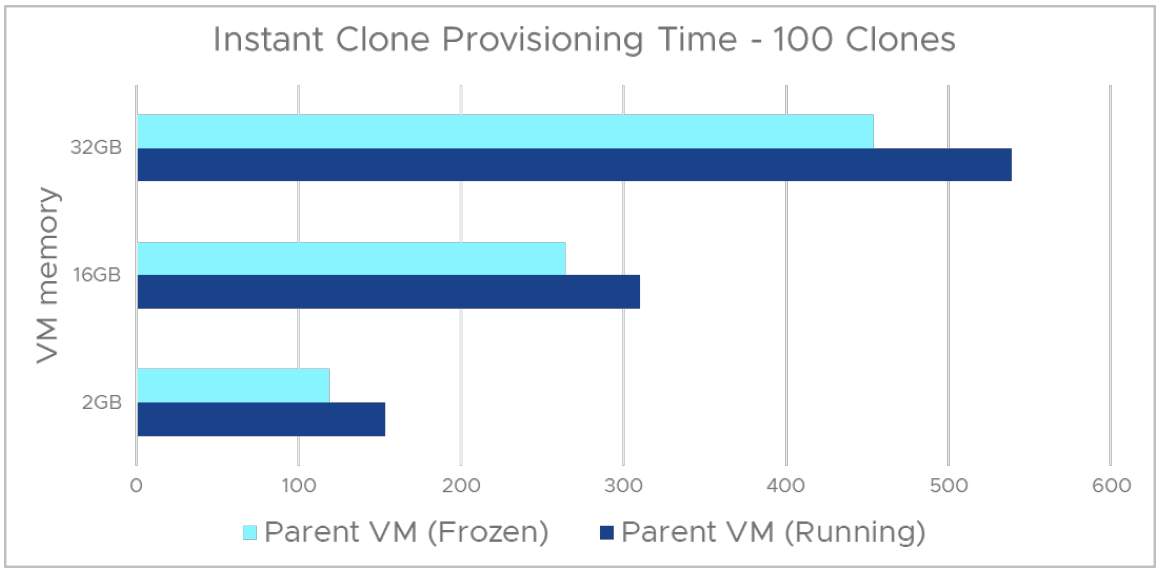

Figure 12: Instant clone creation with different workflows

Figure 12 shows the provisioning duration of 100 instant clones from a running parent VM and a frozen parent VM. In our test scenario, we varied the VM memory size from 2GB to 32GB. In all the scenarios, we observed 15 to 20 percent savings in provision time when using a frozen parent VM.

8.5 Performance of Horizon View with View Planner

In this section, we explore the performance of different types of clones in a virtual desktop deployment with the View Planner benchmark (https://www.vmware.com/products/view-planner.html).

Figure 13 below compares the scalability and performance of a virtual desktop deployment platform with different types of clones for Group A, CPU-intensive applications; and Group B, storage-sensitive applications. The performance metric for View Planner is the quality-of-service (QoS) score, which must be within the 95% percentile of the application response time.

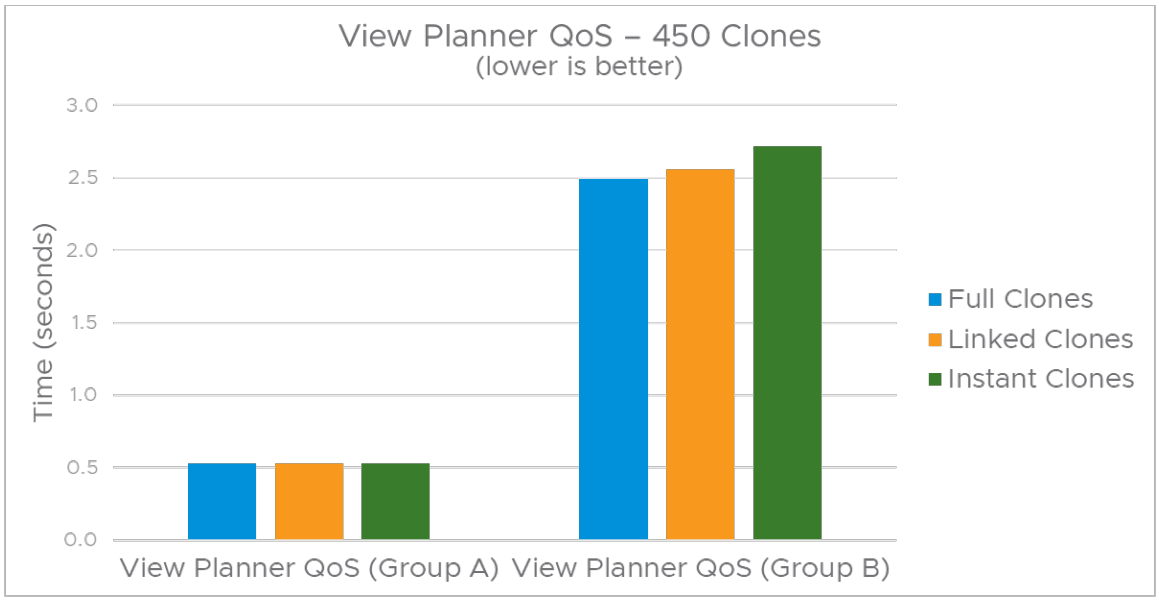

Figure 13: View Planner performance with clones

View Planner workload applications represent those typical for virtual desktop users—such applications include word processors, spreadsheets, web browsers, and document viewers—and are grouped into CPU sensitive (Group A) and storage sensitive (Group B), each with a different QoS requirement. If the QoS scores for both Groups A and B are within the threshold limits provided by View Planner, only then is the test considered to pass. For both Group A and Group B operations, linked clones and instant clones performed very similar to full clones in our VDI test scenario.

9 Clone Performance on vVOL and vSAN datastores

In this section, we explore the performance aspects of clones on different datastores including VMFS, vSAN, and vVOL. To understand the impact of the underlying storage on clone performance, it is helpful to know the variety of snapshot formats supported by VMware. This is because clones are created from a snapshot of the parent VM.

VMware supports multiple different snapshot formats. The type of snapshot format chosen depends on many factors, including the underlying datastore and VMDK characteristics.

vmfsSparse

vmfsSparse, commonly referred to as the redo log format, is the original snapshot format used by VMware. It is the format used on VMFS, NFS (without VAAI-NAS), and vSAN 5.5.

vsanSparse

vsanSparse (https://core.vmware.com/resource/vsansparse-snapshots-tech-note), introduced in vSAN 6.0, is a new snapshot format that uses in-memory metadata cache and a more efficient sparse file system layout, and can operate at much closer base-disk performance levels compared to the vmfsSparse/redo log format.

vVols/native snapshots

In the vVols environment, data services such as snapshot and clone operations are offloaded to the storage array. With vVols, storage vendors use native snapshot facilities, hence vSphere snapshots can operate at a near base-disk performance level.

9.1 Test Environment

VMFS configuration

Dell PowerEdge R930 server:

- 4 sockets, each with a 24-core 2.2 GHz Intel Xeon E7-8890 CPU
- 4TB memory
- Dell EMC Unity 600 all-flash array

vSAN configuration

4-node cluster of Dell PowerEdge R740 servers, each configured with:

- 2 sockets, each with a 20-core 2.4 GHz Intel Xeon Gold 6148 CPU
- 768GB memory
- vSAN version 7.0.2-1.0.17873275
- vSAN datastore - 2 disk groups per host (NVMe cache disk + 2*SATA SSD capacity disk)

vVOL configuration

Dell PowerEdge R740 server:

- 2 sockets, each with a 20-core 2.4 GHz Intel Xeon Gold 6148 CPU
- 768GB memory
- vVOL storage array: Dell EMC PowerStore 5000T (NVMe backed)

9.2 Performance

Table 3 shows a summary of guest application performance loss in the presence of a snapshot with a variety of workloads on all the three datastores including VMFS, vSAN, and vVOL. For each of the datastore scenarios, the performance loss shown with a snapshot is relative to the baseline performance without any snapshot on the respective datastore.

	SPECjbb	Oracle Database /HammerDB	FIO (sequential I/O)	FIO (random I/O)
VMFS	No impact	65% loss	65% loss	65% loss
vSAN	No impact	No impact	No impact	35% loss
vVOL	No impact	No impact	No impact	No impact

Table 3: VMFS is the only datastore that shows a loss in performance for three workloads

The presence of the VM snapshot has the most impact on the guest application performance when using a VMFS datastore. In general, we observe guest applications with a significant disk I/O component losing nearly 65% throughput. In comparison, we observe the presence of VM snapshots on a vSAN datastore has minimal impact on guest application performance for workloads that have predominantly sequential I/O. Of all the three scenarios, the presence of snapshots has the least impact on guest performance when using vVols. In fact, our testing showed the impact was nearly zero even as we increased the snapshot chain.

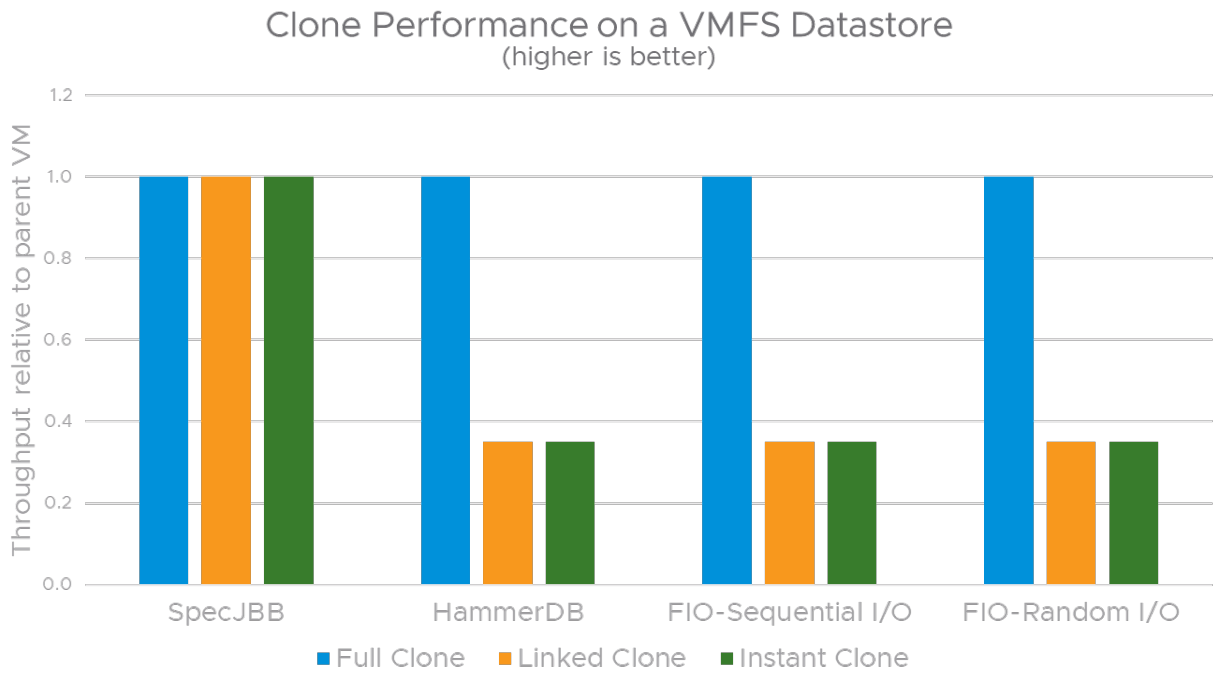

Figure 14: Clone performance when using a VMFS datastore

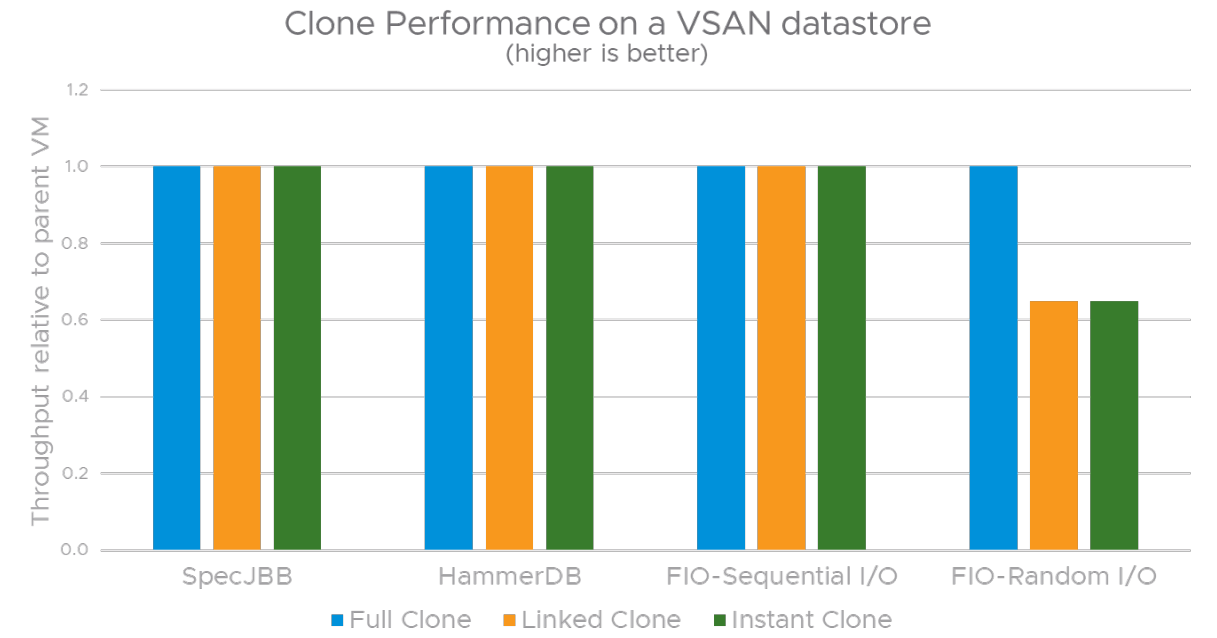

Figure 15: Clone performance when using a vSAN datastore

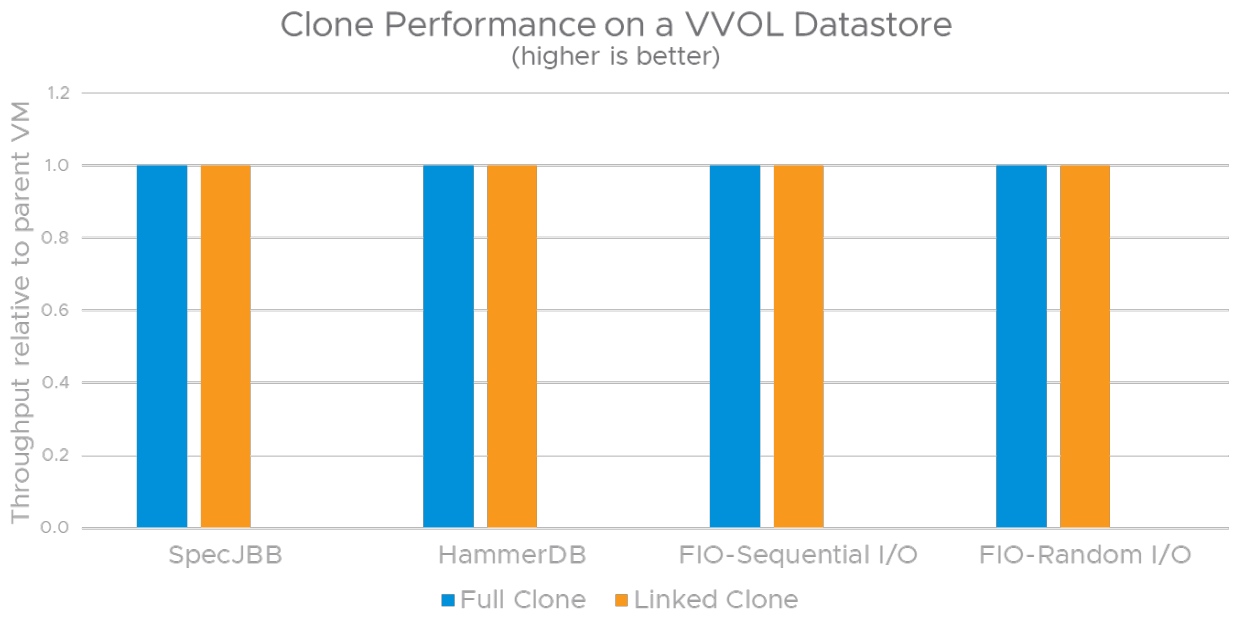

Figure 16: Clone performance when using a vVOL datastore

Figures 14, 15, and 16 show the normalized application performance in a clone (relative to the application performance running inside the parent VM prior to the provisioning operation) with a variety of workloads on VMFS, vSAN, and vVOL datastores. In each of the datastore scenarios, the performance loss seen with instant and linked clones is consistent with the performance loss observed in the presence of VM snapshots. This is because both instant and linked clones use snapshot-based delta disks. Note that vVols do not support instant clones.

The performance data from the wide variety of workloads shows that vVOL is the most performant option when using linked clones and VM snapshots, thanks to the native snapshots supported by the vVOL storage arrays. Using vSAN is a good performance alternative compared to VMFS when using instant clones and VM snapshots.

10 Bulk Provisioning Rates

The tables below provide guidance on the bulk provisioning rates for different VM deployments assuming a VMFS datastore backed by NVMe or SSDs.

Workload	Full Clone	Linked Clone	Instant Clone
Small VMs (for example, web or business ops servers) Config: 4 vCPUs, 16 GB memory, 30GB storage	80 / hour	12,000 / hour	1,400 / hour
Medium VMs (for example, messaging or middleware servers) Config: 8 vCPUs, 32GB memory, 100GB storage	25 / hour	12,000 / hour	800 / hour
Large VMs (for example, traditional relational database servers) Config: 12 vCPUs, 64GB memory, 500GB storage	5 / hour	12,000 / hour	700 / hour

Table 4: Rough estimate on the provisioning rates of different classes of applications

The provisioning time of an instant clone is higher than that of a linked clone because instant clones are always created in a powered-on state, ready for users to connect to with the guest applications running inside the parent VM prior to provisioning, which is a key requirement for some applications. Provisioning an instant clone is a tradeoff of time vs functionality.

As shown in table 4, the provisioning rates of the linked clone and the instant clone are fairly constant and are not impacted by the VM storage size. In contrast, the full clone provisioning rate depends on the total VM storage size and the disk I/O throughput of source and destination datastores. Note that VM storage size refers to actual storage space used by the VM. Unused storage space of thin disks has no impact on the provisioning time. Hence, it might be helpful to express the VM provisioning rate in terms of the storage throughput.

Workload	Full Clone	Linked Clone	Instant Clone
Small VMs (for example, web or business ops servers) Config: 4 vCPUs, 16 GB memory, 30GB storage	40GB/min	—	—
Medium VMs (for example, messaging or middleware servers) Config: 8 vCPUs, 32GB memory, 100GB storage	40GB/min	—	—
750 large VMs (for example, traditional relational database servers) Config: 12 vCPUs, 64GB memory, 500GB storage	40GB/min	—	—

Table 5: Rough estimate of full clone provisioning rates

11 Concluding Remarks

Virtual machine cloning is one of vCenter's most heavily used provisioning operations. vSphere offers three different types of clones—full clones, linked clones, and instant clones—each with differing memory and storage efficiencies and performance. Ultimately, your choice of clone type depends on business goals, workloads, provisioning times, and performance. In this technical paper, we discussed both the performance aspects and provisioning rates of these different clones.

Our tests show the following:

- Full clones generally perform better than linked clones and instant clones because full clones do not share any disk or memory state with a parent VM.
- Instant clones and linked clones are not good candidates for disk I/O–intensive workloads when using a VMFS datastore because they rely on snapshot-based disks and redo logs that result in additional disk I/O latency.
- vSAN and vVOL datastores are a better alternative to a VMFS datastore when using clones or VM snapshots. Instant clones are not supported on vVOL.
- With near identical performance as full clones, instant clones and linked clones are great candidates for Horizon View desktop solutions.
- Beyond Horizon View, instant clones and linked clones perform very well with CPU- and memory-intensive workloads without a significant I/O component.

- Linked clones generally perform slightly better than instant clones because they do not share a memory state with the parent VM.
- The provisioning time of instant clones and linked clones is much shorter than that of full clones.
- The provisioning time of full clones is bound by two factors: total VM storage size and disk I/O throughput of source and destination datastores. The unused storage space of a VM's thin disks has no impact on the provisioning time.
- Provisioning a cold clone (clone from a powered-off parent VM) compared to a hot clone (clone from a running parent VM) increases the vCenter concurrency during bulk clone creation and thereby provides a significant reduction in total provisioning time.
- The provisioning time of linked clones does not vary based on the VM size or disk I/O throughput.
- The provisioning time of instant clones does not depend on disk I/O throughput, although the provisioning time has some dependency on the VM's memory size.
- Provisioning instant clones from a frozen parent VM is helpful both from a deployment standpoint and provisioning time.
- The provisioning time of remote full clones is much slower than that of local full clones.

12 Appendix

The following pyVmomi code example shows how to create an instant clone without any guest customization.

```
import sys
sys.path.append('/usr/lib/vmware/site-packages')

import argparse
import atexit
import getpass
import logging
import ssl

import pyVmomi
import pyVim
import pyVim.connect

from pyVmomi import vim, vmodl
from pyVim.task import WaitForTask

LOG_FORMAT="%(asctime)s %(levelname)s %(name)s:%(funcName)s: %(message)s"
logging.basicConfig(level=logging.DEBUG, format=LOG_FORMAT)
log = logging.getLogger('ictool')


def prompt():

    parser = argparse.ArgumentParser(description='VMware Instant Clone Helper')

    parser.add_argument('--hostname', required=True,           help='vCenter hostname or ip')
    parser.add_argument('--port'    , required=True, type=int, help='vCenter sdk port')
    parser.add_argument('--username', required=True,           help='vCenter username')
    parser.add_argument('--password',                          help='vCenter password')

    parser.add_argument('srcname', help='Name of Source VM')
    parser.add_argument('dstname', help='Name of Destination VM')

    args = parser.parse_args()

    if args.password is None:
        password = getpass.getpass()

    log.debug("%s", args)

    if args.password is None:
        args.password = password

    return args
```

(continued on next page)

```
def get_si(args):
    log.info("Attempting to connect to %s:%s", args.hostname, args.port)
    si = pyVim.connect.SmartConnect(host=args.hostname,
                                    user=args.username,
                                    pwd=args.password,
                                    port=args.port,
                                    sslContext=ssl._create_unverified_context())
    atexit.register(pyVim.connect.Disconnect, si)
    if not si:
        log.error("Unable to connect to %s:%s", args.hostname, args.port)
        sys.exit(0)

    log.info('Successfully connected %s', si)
    args._si = si


def get_vms(args):
    log.debug("Querying all VMs")
    content = args._si.content
    view = content.viewManager.CreateContainerView(content.rootFolder, [vim.VirtualMachine],
True)
    vmlist = view.view
    view.Destroy()
    args._vmlist = vmlist
    log.debug("Got %s VMs", len(vmlist))


def find_vm(args, name):
    if not hasattr(args, '_vmlist'):
        get_vms(args)

    for vm in args._vmlist:
        if vm.name == name:
            return vm
    log.error("Unable to find VM named %s", name)
    sys.exit(1)
```

(continued on next page)

```
def main():

    args = prompt()
    si = get_si(args)

    srcvm = find_vm(args, args.srcname)

    log.info("Found %s as %s", args.srcname, srcvm)

    icspec = vim.vm.InstantCloneSpec()
    icspec.location = vim.vm.RelocateSpec()
    icspec.name = args.dstname

    log.info("InstantCloning %s to %s with spec %s",
             args.srcname, args.dstname, icspec)

    ictask = srcvm.InstantClone(icspec)
    WaitForTask(ictask)

    dstvm = ictask.info.result

    log.info("Resulting vm %s is %s", args.dstname, dstvm)

if __name__ == '__main__':
    main()
```

(end of code example)

About the Author

Sreekanth Setty is a staff member with the performance engineering team at VMware. His work focuses on investigating and improving the performance of VMware's virtualization stack, most recently in the live-migration and clone space. He has published his findings in many technical papers and has presented them at various technical conferences. He has a master's degree in computer science from the University of Texas, Austin.

Acknowledgements

The author sincerely thanks Janitha Karunaratne and Tanmay Nag for their contributions to the paper. He also would like to extend thanks to the members in his team including Yury Baskakov, Ying Yu, Nick Shi, Ruban Deventhiran, Tony Lin, and Julie Brodeur for their reviews.