OpenVMS Technical Journal V19

Best practices of using SSD

Table of contents

HP solid state drives (SSDs) deliver exceptional performance for customers with applications requiring high random read IOPS performance.

Based on single-level cell (SLC) or multi-level cell (MLC) NAND flash technology, the SSDs support HP ProLiant servers and server storage platforms. They are available as Small Form Factor (SFF) and Large Form Factor (LFF) hot plug devices, non-hot plug SFF devices, and SFF Quick Release devices.

HP SSDs deliver higher performance, lower latency, and lower power solutions when compared with traditional rotating media and fit seamlessly into the existing HP Integrity server infrastructure.

OpenVMS supports SSDs starting from OpenVMS V8.4 Update 700. SSD drives in OpenVMS are supported in both HBA and RAID mode with a SAS controller. HBA mode provides direct access to SSD devices as a pass-through module, and with RAID we can take advantage of SAS Controller to carve RAID disks, of multiple SSD devices.

Engineering community has done experiments and came up with the best practices for using an SSD. For example, SSD gives better performance when there are many parallel random reads than when there is one large sequential read IO. This article discusses some of the important aspects with SSD which should help user to choose between lifetime or endurance, cost of the disk, and the performance.

Performance benchmarking of an SSD

OpenVMS engineering has done various performance tests with SSD. We have also done testing with traditional SAS hard disks to compare the results. The tests conducted were small block random read or write which simulate OLTP workloads and large block sequential read or write which simulate a media or backup workload.

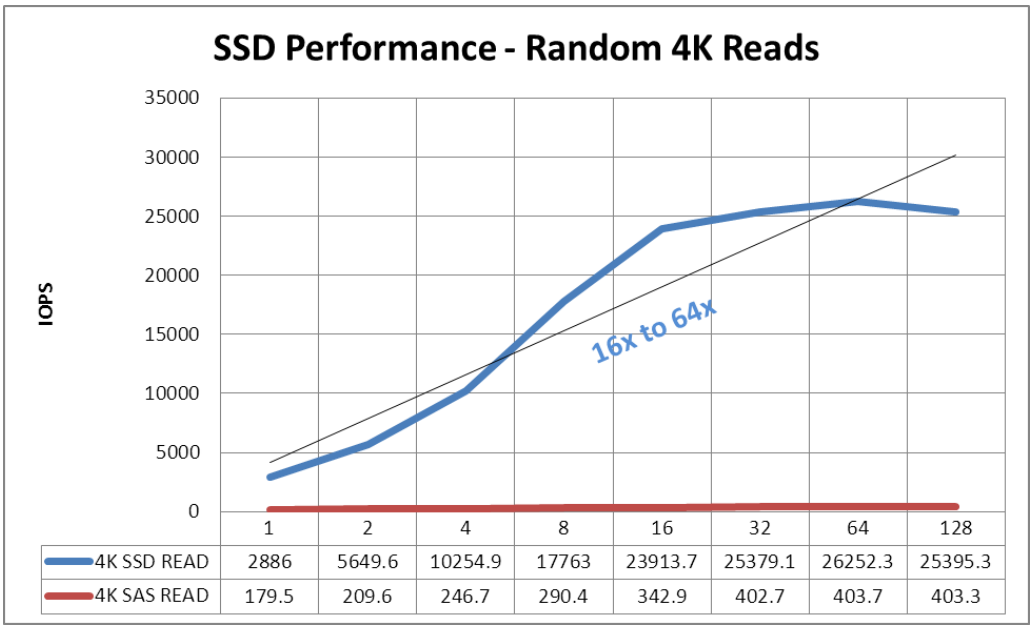
Random reads

For a traditional HDD to perform an IO, it requires the rotation or movement of magnetic head. An SSD does not possess a magnetic head that a traditional disk uses to perform the IO. Instead, it uses electronic technology to read the data. The performance of an IO is highly dependent on drive latency or disk access time. The disk access time is sum of drive-seek time, rotational delay, and transfer delay. As mentioned, with SSDs, there is neither seek time nor rotational delay, resulting in faster access time. Thus a random read SSD gives a very high degree of performance over a traditional HDD. Note that random reads are much faster compared to writes due to slowness of flash programming operations (write).

Figure 1 depicts the result of a test where random read IO performance of SSD and traditional SAS HDD are compared. Number of threads are measured along X-axis (horizontal) and number of random read IOPS are measured along Y-axis (vertical). Each thread performs a random read IO of size 4K.

From the graph, we can see that SSD gives a very high degree of performance (16X to 64X) over traditional SAS HDD when there is huge number of random read IOs.

Figure 1. Performance of SSD v/s traditional HDD for Random Reads



	1	2	4	8	16	32	64	128
4K SSD READ	2886	5649.6	10254.9	17763	23913.7	25379.1	26252.3	25395.3
4K SAS READ	179.5	209.6	246.7	290.4	342.9	402.7	403.7	403.3

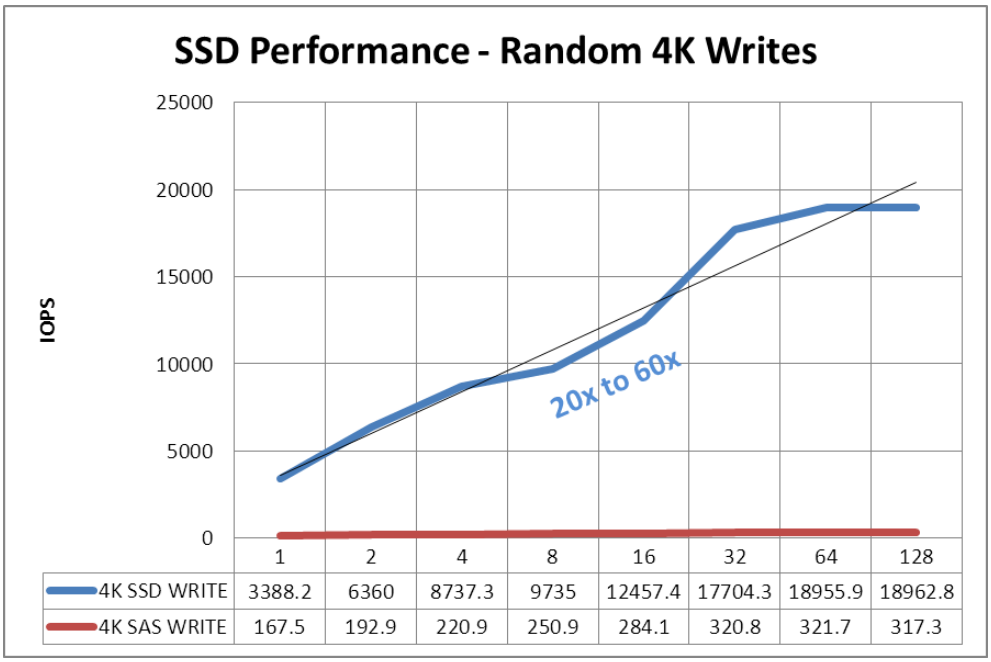
Random writes

In a traditional HDD, to execute random writes the overhead of the magnetic head movement exists as in the case of random read. So similar to random reads, SSD gives a very high degree of performance over a traditional HDD for random writes as well.

Figure 2 depicts the result of a test where random write IO performance of SSD and traditional SAS HDD are compared. Number of threads are measured along X-axis (horizontal) and number of random write IOPS are measured along Y-axis (vertical). Each thread performs a random write IO of size 4K.

From the graph, SSD gives a performance gain of around 20X to 60X over traditional SAS HDD for random write IOs.

Figure 2. Performance of SSD v/s traditional HDD for Random Writes

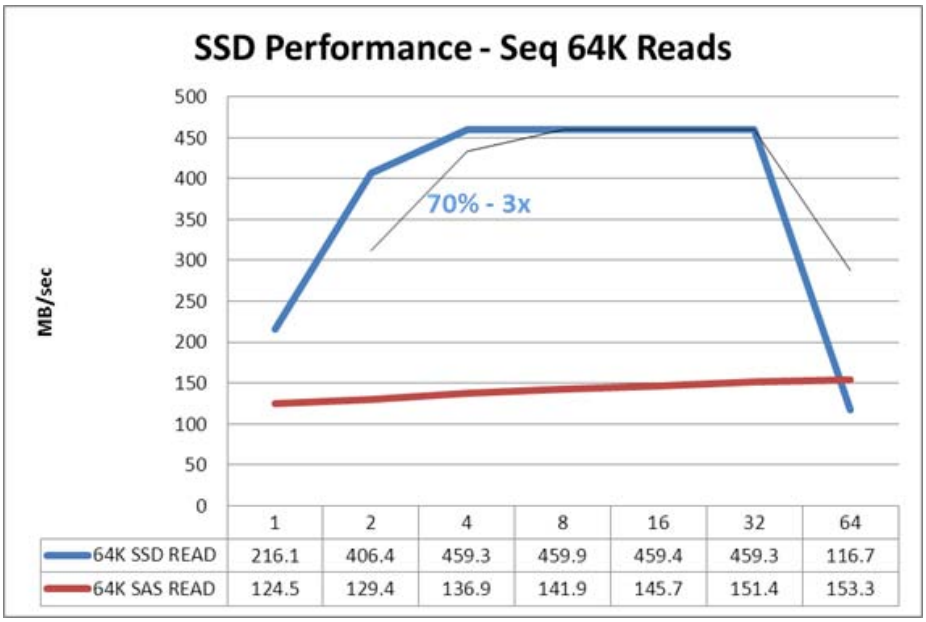


Sequential reads

Figure 3 depicts the result of a test where sequential read IO performance of SSD and traditional SAS HDD are compared. Number of threads are measured along X-axis (horizontal) and number of read IOPS are measured along Y-axis (vertical). Each thread performs a sequential read IO of size 64K.

From the graph, we can see that SSD gives a performance around 3X over traditional SAS HDD when there is huge number of sequential read IOs.

Figure 3. Performance of SSD v/s traditional HDD for Sequential Reads



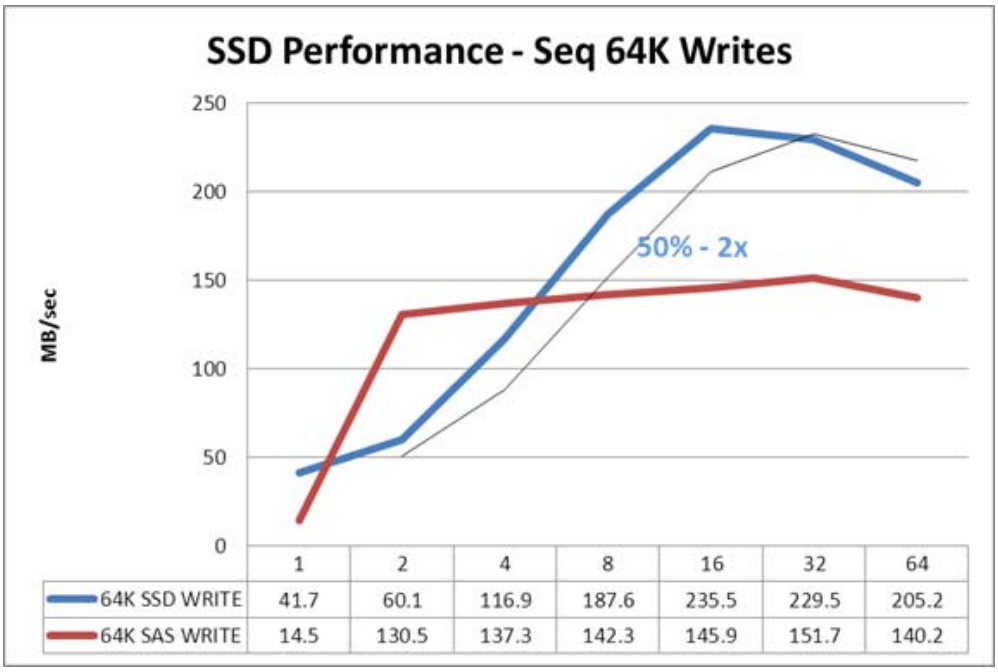
Sequential writes

For a traditional HDD, the sequential write performance is better than random write as in the case with sequential reads. Similarly, in this case also, SSD provides significant performance improvement over a traditional HDD for sequential writes.

Figure 4 depicts the result of a test where sequential write IO performance of SSD and traditional SAS HDD are compared. Number of threads are measured along X-axis (horizontal) and number of write IOPS are measured along Y-axis (vertical). Each thread performs a sequential write IO of size 64K.

From the graph, we can see that SSD gives a performance around 2X over traditional SAS HDD when there is huge number of sequential write IOs.

Figure 4. Performance of SSD v/s traditional HDD for Sequential Writes



To summarize the test results, SSD provides approximately 2X performance improvement over a traditional SAS disk on sequential IO. For a random IO, SSD gives much better performance improvement in the range of 15X to 60X.

SSD lifetime

An SSD's lifetime depends on number of writes performed to it. We can find out the remaining lifetime of an SSD using the lexical F$GETDVI with item code SSD_USAGE_REMAINING (based on current workload in percentage) and SSD_LIFE_REMAINING (estimated number of days left as per current workload). SSD has to be connected locally for this lexical to work.

For example:

```
$ SSD_USAGE_REMAINING=F$GETDVI("DKA100:","SSD_USAGE_REMAINING")
$ SHOW SYMBOL SSD_USAGE_REMAINING
SSD_USAGE_REMAINING = 100 Hex = 00000064 Octal = 00000000144

$ SSD_LIFE_REMAINING = F$GETDVI("DKA100:","SSD_LIFE_REMAINING")
$ SHOW SYMBOL SSD_LIFE_REMAINING
SSD_LIFE_REMAINING = 610372 Hex = 00095044 Octal = 00002250104
```

The new item codes DVI$_SSD_USAGE_REMAINING and DVI$_SSD_LIFE_REMAINING are available for SYS$GETDVI system service, LIB$GETDVI library routine and F$GETDVI lexical function.

MSA$UTIL also has been enhanced to display the detailed SSD gas gauge information.

For more details on the usage of the gas gauge features, please refer to OpenVMS V8.4 Update 700 release notes or online DCL help.

SSD and XFC cache

Enabling XFC (Extended File cache, which is file system data cache for OpenVMS) cache with SSD boosts the performance. Having XFC cache gives performance improvements of approximately 2X to 5X on SSDs for sequential read IO when compared to XFC-disabled scenario.

This is because RAM is much faster than NAND-based SSD device. Thus using XFC cache will boosts read performance. Please note that since XFC is write-through cache there won't be any performance gain with XFC for write IO.

RMS-deferred write

Since SSD's lifetime is dependent on number of writes, it is good if we reduce the number of writes to an SSD. The number of writes can be reduced by using RMS-deferred write. If RMS-deferred write is enabled on a file, then the records written to the file are kept in RMS local buffers, and the final write to the disk are done once the RMS buffers are full or a $Flush or $close is issued on the file.

The buffer size used for RMS deferred write can be decided by the value of multi block count (MBC) and multi buffer count (MBF). MBC is the size of the buffer used by RMS in blocks and MBF is number of buffers used by RMS.

We can set process or system default value for MBC and MBF using SET RMS command.

For example:
For setting them process-wide,

```
$ SET RMS/BLOCK_COUNT=100/BUFFER_COUNT=10
```

Here MBC is set to 100 and MBF is set to 10.

For setting them system-wide, use /SYSTEM qualifier with the above command.

Default value of MBC is 32 and it can be controlled using the SYSGEN parameter RMS_DFMBC.

Consider the following example. The indexed file has a single key and its records are 100 bytes long. The bucket size is three blocks with a fill factor of 67 percent. Thus, there is an average of 10 records in each bucket. A batch program reads each record and updates part of it, beginning at the first record in the file and moving through the records sequentially. Without the deferred-write option, 11 disk I/O operations occur for every 10 records—one to read the bucket and one to write the bucket for each record. With the deferred write option, only two disk I/O operations occur for every 10 records—one to read the bucket and one to write the bucket after the record operations are completed.

For more details on RMS-deferred write, please refer "OpenVMS Record Management Services Reference Manual" and "Guide to OpenVMS File Applications".

SSD in a volume set

SSD can be used in a volume set along with any other disk. We can increase the cost-effective performance by creating a volume set consisting of an SSD and a traditional hard disk. One use case of this approach would be to use the volume set to store RMS-indexed files. Here the index area, which consumes lesser space, can be stored on SSD and the data area on the traditional hard disk. This might speed up the access of indices of the file by RMS.

Please note that, there are overheads in managing volume sets.

For example, To create a volume set of an SSD and a traditional SAS disk,

```
$ MOUNT/BIND=SSD_SAS_VOL_SET DGA0:,DGA1: SSD_DISK SAS_DISK
```

Using SSDs with files of high XFC cache hits

A file which has higher number of cache hits are the most important or frequently accessed files on the system. Customer can choose to store those files on SSDs, so that the read performance may improve.

To figure out the number of cache hits or misses, customer can use the command,

```
$ SHOW MEMORY/CACHE=FILE=<absolute file path>

E.g. $ SHOW MEMORY /CACHE=FILE=SYS$SYSDEVICE:[TEST.INDEX]index.c
$ SHOW MEMORY/CACHE=(VOLUME=<Volume_Name>,OPEN,CLOSED,TOPQIO)

E.g. $ SHOW MEMORY/CACHE=(VOLUME=SYS$SYSDEVICE:,OPEN,CLOSED,TOPQIO)
```

Conversely, customer may also choose to store the files on SSD which have higher number of cache misses since those are the files which can be read from the disks. Storing those files on SSDs rather than on traditional hard disks may boost the performance.

Defragmenter on SSD

On a traditional disk, if the files are randomly arranged it can lead to performance degradation because random read is costly. Whereas, an SSD gives same performance for random reads and sequential reads. So there is no need to use a defragmenter on SSD.

But, if the file is highly fragmented, the file can have many extents. Highly fragmented file introduces some overhead in file system processing as different extents need to be handled and several internal data structures have to be traversed. It may result in small performance degradation. For a single read on a large and highly fragmented file, we got a performance degradation of around 5 percent in our tests. Users can increase the cluster factor or pre-allocate the space to files on the SSD to lower this impact. Alternatively, mounting the disk with higher window size can also help here.

Another impact of highly fragmented files are that, some of the files in OpenVMS require contiguous space. In case of not running defragmenter, there is a chance that the disk would be left highly fragmented and there may not be contiguous space for files which require contiguous allocation. In such a case if customer notices any performance degradation, we would recommend the customer to run defragmenter on the SSD.

Disabling high water marking

Lifetime of an SSD can be improved by disabling high watermarking. When a user deletes the file the contents of the file will not be erased if the high watermarking is disabled on the disk, thus reducing number of writes to the disk. But, it can have an impact on the data security. Users have to consider the aspects of security and performance while taking a decision on this.

For example, to enable high-water marking on the disk,

```
$ SET VOLUME /HIGHWATER_MARKING <DISK_NAME>
```

To disable high-water marking on the disk,

```
$ SET VOLUME /NOHIGHWATER_MARKING <DISK_NAME>
```

Learn more

For all information about the best practices, visit
http://h71028.www7.hp.com/enterprise/us/en/technologies/solid-state-storage-overview.htm

Share your feedback or queries on the VMS Technical Journal here

Sign up for updates
hp.com/go/getupdated

Rate this document

4AA4-4794ENW, January 2013