Storage Performance Guidelines

First of all, StorPerf is not going to give you pointers on how to tune your Cinder implementation, as there are far too many backends (Ceph, NFS, LVM, etc), each with their own methods of tuning. StorPerf is here to assist in getting a reliable performance measurement by encoding the test specification from SNIA, and helping present the results in a way that makes sense.

Having said that, there are some general guidelines that we can present to assist with planning a performance test.

Workload Modelling

This is an important item to address as there are many parameters to how data is accessed. Databases typically use a fixed block size and tend to manage their data so that sequential access is more likely. GPS image tiles can be around 20-60kb and will be accessed by reading the file in full, with no easy way to predict what tiles will be needed next. Some programs are able to submit I/O asynchronously where others need to have different threads and may be synchronous. There is no one size fits all here, so knowing what type of I/O pattern you need to model is critical to getting realistic measurements.

System Under Test

The unfortunate part is that StorPerf does not have any knowledge about the underlying OpenStack itself – we can only see what is available through OpenStack APIs, and none of them provide details about the underlying storage implementation. As the test executor, you will need to know information such as: the number of disks or storage nodes; the amount of RAM available for caching; the type of connection to the storage and bandwidth available.

Measure Storage, not Cache

As part of the test data size, you need to ensure that you are preventing caching from interfering in your measurements. The total size of the data set in the test must exceed the total size of all the disk cache memory available by a certain amount in order to ensure we are forcing non-cached I/O. There is no exact science here, but if we balance test duration against cache hit ratio, it can be argued that 20% cache hit is good enough and increasing file size would result in diminishing returns. Let's break this number down a bit. Given a cache size of 10GB, we could write, then read the following dataset sizes:

- 10GB gives 100% cache hit
- 20GB gives 50% cache hit
- 50GB gives 20% cache hit
- 100GB gives 10% cache hit

This means that for the first test, 100% of the results are unreliable due to cache. At 50GB, the true performance without cache has only a 20% margin of error. Given the fact that the 100GB would take twice as long, and that we are only reducing the margin of error by 10%, we recommend this as the best tradeoff.

How much cache do we actually have? This depends on the storage device being used. For hardware NAS or other arrays, it should be fairly easy to get the number from the manufacturer, but for software defined storage, it can be harder to determine. Let's take Ceph as an example. Ceph runs as software on the bare metal server and therefore has access to all the RAM available on the server to use as its cache. Well, not exactly all the memory. We have to take into account the memory consumed by the operating system, by the Ceph processes, as well as any other processes running on the same system. In the case of hyper-converged Ceph, where workload VMs and Ceph run on the systems, it can become quite difficult to predict. Ultimately, the amount of memory that is left over is the cache for that single Ceph instance. We now need to add the memory available from all the other Ceph storage nodes in the environment. Time for another example: given 3 Ceph storage nodes with 256GB RAM each. Let's take 20% off to pin to the OS and other processes, leaving approximately 240GB per node This gives us 3 x 240 or 720GB total RAM available for cache. The total amount of data we want to write in order to initialize our Cinder volumes would then be 5 x 720, or 3,600 GB. The following illustrates some ways to allocate the data:

- 1 VM with 1 3,600 GB volume
- 10 VMs each with 1 360 GB volume
- 2 VMs each with 5 360 GB volumes

Back to Modelling

Now that we know there is 3.6 TB of data to be written, we need to go back to the workload model to determine how we are going to write it. Factors to consider:

- **Number of Volumes.** We might be simulating a single database of 3.6 TB, so only 1 Cinder volume is needed to represent this. Or, we might be simulating a web server farm where there are hundreds of processes accessing many different volumes. In this case, we divide the 3.6 TB by the number of volumes, making each volume smaller.
- **Number of Virtual Machines.** We might have one monster VM that will drive all our I/O in the system, or maybe there are hundreds of VMs, each with their own individual volume. Using Ceph as an example again, we know that it allows for a single VM to consume all the Ceph resources, which can be perceived as a problem in terms of multi-tenancy and scaling. A common practice to mitigate this is to use Cinder to throttle IOPS at the VM level. If this technique is being used in the environment under test, we must adjust the number of VMs used in the test accordingly.
- **Block Size.** We need to know if the application is managing the volume as a raw device (ie: /dev/vdb) or as a filesystem mounted over the device. Different filesystems have their own block sizes: ext4 only allows 1024, 2048 or 4096 as the block size. Typically the larger the block, the better the throughput, however as blocks must be written as an atomic unit larger block sizes can also reduce effective throughput by having to pad the block if the content is smaller than the actual block size.
- **I/O Depth.** This represents the amount of I/O that the application can issue simultaneously. In a multi-threaded app, or one that uses asynchronous I/O, it is possible to have multiple read or write requests outstanding at the same time. For example, with software defined storage where there is an Ethernet network between the client and the storage. The storage would have a higher latency for each I/O, but is capable of accepting many requests in parallel. With an I/O depth of 1, we spend time waiting for the network latency before a response comes back. With higher I/O depth, we can get more throughput despite each I/O having higher latency. Typically, we do not see applications that would go beyond a queue depth of 8, however this is not a firm rule.

- **Data Access Pattern.** We need to know if the application typically read data sequentially or randomly, as well as what the mixture of read vs. write is. It is possible to measure read by itself, or write by itself, but this is not typical behavior for applications. It is useful for determining the potential maximum throughput of a given type of operation.

Fastest Path to Results

Once we have the information gathered, we can now start executing some tests. Let's take some of the points discussed above and describe our system:

- OpenStack deployment with 3 Control nodes, 5 Compute nodes and 3 dedicated Ceph storage nodes.
- Ceph nodes each have 240 GB RAM available to be used as cache.
- Our application writes directly to the raw device (/dev/vdb)
- There will be 10 instances of the application running, each with its own volume.
- Our application can use block sizes of 4k or 64k.
- Our application is capable of maintaining up to 6 I/O operations simultaneously.

The first thing we know is that we want to keep our cache hit ratio around 20%, so we will be moving 3,600 GB of data. We also know this will take a significant amount of time, so here is where StorPerf helps.

First, we use the configurations API to launch our 10 virtual machines each with a 360 GB volume. Next comes the most time consuming part, we call the initializations API to fill each one of these volumes with random data. By preloading the data, we ensure a number of things:

- The storage device has had to fully allocate all of the space for our volumes. This is especially important for software defined storage like Ceph, which is smart enough to know if data is being read from a block that has never been written. No data on disk means no disk read is needed and the response is immediate.
- The RAM cache has been overrun multiple times. Only 20% of what was written can possibly remain in cache.

This last part is important as we can now use StorPerf's implementation of SNIA's steady state algorithm to ensure our follow up tests execute as quickly as possible. Given the fact that 80% of the data in any given test results in a cache miss, we can run multiple tests in a row without having to re-initialize or invalidate the cache again in between test runs. We can also mix and match the types of workloads to be run in a single performance job submission.

Now we can submit a job to the jobs API to execute a 70%/30% mix of read/write, with a block size of 4k and an I/O queue depth of 6. This job will run until either the maximum time has expired, or until StorPerf detects steady state has been reached, at which point it will immediately complete and report the results of the measurements.

As a closing note, StorPerf does use FIO as its workload engine, so whatever workload parameters we would like to use with FIO can be passed directly through via StorPerf's jobs API.