CS580K : Shubham Patwa
Written Assignment 1

1. What is "dual I/O stacks" in storage virtualization? What is the consequence of dual I/O stacks? What are the solutions you can come up to mitigate this problem?
Solution:
Dual I/O Stacks:
If we issue an I/O request from the application running inside the VM , it goes through the following:
I/O stack in the VM -> I/O stack in the native host.
Consequence :
Results in long I/O path and long I/O latency.
Solution :

1. Simplify the I/O stack in VM by using a thin para virtualisation layer. We call it the Front I/O end. Its goal is to communicate the I/O requests to the hypervisor instead of going through multiple I/O layers.

Result : Using this, we can eliminate a lot of I/O layers in the VM such as I/O schedulers,etc

2. Shorten the I/O path :Either by creating a short path between the VM front end and the emulator. Usually QEMU serves as an emulator or completely placing the emulator inside the native host.

Result : I/O request can be quickly delivered to the native and the native I/O stacks can then handle the request.

3. Leverage hardware assisted technology : Such as VTD,etc They will designate one device or part of one device for using virtual functions to a VM.

Result : This way it can bypass the whole native host and can achieve the same performance or as near as the native, since it can directly access the underlying I/O devices without going through the I/O stack in the native host.

2. Why do we need a "byte-addressable" storage stack in cloud block storage system? What problem does tradition block-addressable storage cause in cloud block storage system, and how does the byte-addressable storage stack solve them?
Solution:
Problems:

1. **Long I/O latency :**

For non -aligned writes, block addressable always aligns the nonaligned I/O writes with the page cache size for ex : 4k bytes. The FS will fetch the data from the storage backend to the local page cache and then puts the new writes to the corresponding slot of the page cache because in such a multilayered cloud storage system, fetching data from storage backend is time consuming. It results in long I/O latency due to data granularity.

2. **Ineffiecint use of storage** :When the FS needs to flush dirty data from the page cache to the backend storage devices: Due to block addressability the FS will flush the whole data of one page i.e 4k regardless of how many bytes are actually dirtied. It causes inefficient use of storage.

Solution :
Have a byte addressable storage stack.

1. By redesigning the whole I/O stack including the client side and the server side, byte addressable storage stack can generate arbitrary length I/O request, hence achieving byte addressability.
2. We redesign the page cache by enabling byte addressability to keep track to application level access at a byte granularity through which we know how many bytes are dirtied and flush only those.

3. There is a saying that "every write for SSD" is a random write. Do you agree or not? Why?

- Yes , I agree that every write is random in SSD.
- For a write in SSD, it is out-of-place write.
- It means that it writes to a clean page and marks it as valid and thus marks the original page as invalid.
- That is we cannot simply write to an SSD, we have to first erase and then write the data.
- When a block is chosen to erase data , valid pages are moved to another block. Then , invalid pages are erased in the block. Once it is erased, data will be written.
- Since, we cannot write directly, it chooses any random block for write.

4. In object storage, if we use a "simple" hashing algorithm, to add/remove a storage device would lead to high overhead (i.e., data will be moved around devices). The consistent hashing algorithm is then used to solve the problem. Please explain how the consistent hashing algorithm works.
Solution:
Problem :When we want to add/remove drives in simple hashing algorithm:
Result : The hash values of all objects will stay the same, but we will need to recompute the mapping value for all objects and then remap them to the different drives.
This operation is very expensive. In the worst case, all objects will be moved from one location to another location.
Solution :
Consistent Hashing Algorithm:
Instead of generating hashing values for all the objects, each drive will be assigned a range of hash values to store the objects.
For example, suppose the drives are assigned the range :
Drive 0 - 0000 ~ 3ffe
Drive 1 - 3fff ~ 7ffe
Drive 2 - 7fff ~ bffe

Drive 3 - bffe ~ efff
Mapping of objects to different drives

Image 1 : hash value -> b5e7….	Drive mapped to drive 2
Image 2: hash value -> 9433…	Drive mapped to drive 2
Image 3: hash value -> 1213…	Drive mapped to drive 0

..
Now when we add a new drive, each drive will get a new range of hash values it is going to store.
But still, each object's hash value will still remain the same.
Now, the moving of objects will depend on the new range of hash values of the drive. For objects whose hash value is not within range of its current drive will be mapped to another drive.
But number of objects to be moved is very few using consistent hashing algorithm, compared to basic hash function.

5. What are the pros and cons of object storage in comparison with block storage? Especially, what makes objects storage much scalable?

Pros :

1. It is compatible with unstructured work loads due to its flat structure,it doesn't have same limitations as block storage.

2. Scaling: the system scales out by simply adding more nodes. It can scale to terabytes without any restrictions.
3. Direct access to individual objects, no need to transverse directories.
4. Ideal for storing cool/cloud data.

5. Software centric, data durability

Cons:

1. Not suitable for hot data : Due to its unstructured characteristics object storage does not allow you to alter just a price of a data block, you must read and write an entire object at once.Eg., in the block storage system , we can easily append a line in a file but in an object storage system , you need to retrieve the object , add a new line and write the entire object back. This makes object storage less ideal for data that change very frequently(hot data) especially when the object size is very huge, to update a tiny portion of it, you need to update the whole object.

What makes object storage scalable?
The flat structure of object storage makes it scalable by simply adding more nodes. Devices can be added by using hashing algorithms such as consistent hashing algorithm.