1. **Describe the benefits of virtualization for cloud computing.**

Virtualization refers to technique used to run several instances of computer system in a layer which is abstracted from actual hardware. It is made possible through OS like software called hypervisors that runs on top of hardware and can run several Virtual Machines on top of it. However, these days, meaning of virtualization has grown. Cloud computing do not just use virtualization in case of Virtual Machines. They provide virtualization in different levels. For instance: now we can virtualize using containers. Here we do not need to worry about what OS to choose that is OS is virtualized. In recent time, cloud computing has climbed one more level in virtualization by virtualizing servers itself with serverless where user can deploy their functions or lambdas. Now let’s discuss some of the benefits that we can get from the virtualization in cloud computing:

1. **Management-free:** Had we not have any virtualization; we would be required to buy actual hardware to run and deploy the application we want. We would have to first buy hardware, then set up OS or VMM before we can run the application of our choice. That can be very daunting especially for the beginner. In the world of serverless, management gets even easier; all we need to do is write our business logic. We do not need to be in the situation where the server keeps on crashing for no reason. We can simply deploy our application on top of FaaS and let it handle failure, replication, fault tolerance, etc.
2. **Flexible:** Had we not have virtualization within cloud computing, we would not be able to easily switch between different hardware of our choice. Once we buy a hardware, we would not be able to upgrade it very easily. However, because of virtualization we can simply upgrade to better computing resource with press of a bottom.
3. **Autoscaling:** Especially in modern virtualization like serverless, the application will automatically be scaled up and down according to the demand of the application. This would not be true if we actually own hardware. We would have to add more hardware or upgrade to powerful devices to scale up. And what happens when we need to scale down? That’s where cloud computing saves our day.
4. **Paying for what we use:** When we just deployed our first application, we will probably not have much traffic. If it weren’t for virtualization, we would be paying for entire hardware upfront. Because of virtualization, we are now able to run our application almost at no or negligible cost.
5. **In a cluster of 12 CPU cores and 48 GB of memory, one job asks a resource scheduler to launch tasks of 1 CPU core and 8 GB of memory, and another job asks to launch tasks of 4 CPU core and 2 GB of memory. What are the dominant resources of such two tasks? What are their dominant shares? With the dominant resource fairness, how many tasks can be launched in the cluster for the two jobs?**

Dominant resource of the 1st task would be memory since it requests for 16.66% (8/48) of the available memory and only 8.33% (1/12) cores of CPU, while the dominant resource of the 2nd task will be the CPU since it requests 33.33% (4/12) cores of the CPU and only 4.166% (2/48) of the total memory to run its task.

Therefore, dominant shares for task 1 would be 16.66% on memory and task 2 would be 33.33% on CPU cores.

Now if we apply dominant resource fairness, which means we try to equalize the dominant share for each of the jobs.

We do so by giving more than 50% for each job’s dominant resource.

For Job 1: We can run **4 tasks** which would take **66.6%** of the memory and take **33.32%** of CPU.

For Job 2: We can run **2 tasks** which would take **66.6%** of the CPU and take **8.32%** of memory.

1. **Why is the task with a finer granularity preferred by a cloud-scale resource scheduler?**

In order to achieve high scalability and efficiency, every task needs to be designed in such a way that it will take short amount of time and requires only a fraction of resources.  That way there will never be situation where shorter task have to keep waiting on the queue to be scheduled since some enormous task may take forever to complete. For instance: we might have a task that will take several hours to finish. During that period of time, there might be other jobs that only take several minutes waiting in the queue to be scheduled. It is not really fair for a task to wait several hours only to get some minutes of resources. That is why we need to break big task in such a way that it has finer granularity. If we have a task that takes several hours we need to try to break it into hundreds of smaller tasks so that overall each of the broken tasks will take only a few minutes and require much less resources.

Finer granularity also gives us the following benefits:

**Improvement in data locality:** If the task is big then the input is also probably large. This means the required data might not be located on a single server. That will be a problem since I/O is always much slower. Therefore, by dividing our task in finer granularity, all our task can be run in the server that happens to have the required small chunk of input data.

**Easier to handle node failures:** When each task is small, with high probability we can say that it doesn't meet any node failure during its running period. Even if it does, we can easily migrate that task to other running nodes. However, it would not be so easy if our task was large. It would take much longer for the whole task to be migrated to different nodes and our progress will also be lost while this migration takes place.

1. **Why stateful applications are bad cases for serverless computing?**

Stateful applications are bad cases for serverless computing because serverless function will not by itself preserve any state between different invocation. This is because the function by its nature is meant to be mobile that is, it should be able to run in one VM for now and should be able to migrate to a different VM later on. The application cannot possibly take the state with it since states are usually saved in the RAM which will certainly be gone once the function is served from a different server. One way to preserve the state would be to use a database but it does not really make sense to query the database for the state of application since databases are too slow. Therefore, if our applications need to preserve state from one invocation to another, we probably should not be using serverless computing at all.

1. **What is consistent hashing and why is it desirable for the distributed hash table?**

Consistent hashing solves the problem of master or directory server being so large that we might have to make it distributed. It is very desirable in case of distributed hash table because it provides balance because no bucket will have too many loads and it also gives smoothness to the system since addition and removal of servers won’t change the hashing function like regular hashing with modulo would and therefore movement of key value will be minimized by a lot while also maintaining the uniformity or spread of key values across available servers. We can also have a great storage fault tolerance using this technique. We can simply store a replica of a key into the next K servers that we see in the circle of hashes. Now when one of those K servers goes down, we know for sure the key value is still stored in the same partition but maybe on a different server.

Basically, what consistent hashing does is, it associates each node with a unique id in an uni-dimensional space 0...2^(m-1). Then the space is partitioned across M servers. Now each key value will be stored at the server that has the smallest ID larger than the hash of the key. This technique basically makes the central master or directory server obsolete (for instance: if we use DHT Chord Basic Lookup) which is a very nice thing in the world of distributed computing.

1. **How does hinted handoff work in Dynamo?**

Hinted handoff is a technique used by Coordinator nodes in Dynamo Key Value Store. When one or many of the nodes in the coordinator's preference list is down and the total replies received while performing put() is less than W (W<N), the replica might be forwarded beyond to the node falling outside the replication factor (N). But the replica that the node outside N receives will contain a “hint” that it should actually be forwarded to the node that was down. Therefore, the node that gets the replica will periodically try forwarding the replica to the currently down node. When the node comes back up, the forwarding will be done and the node that was down receives the replica that it should have received from the coordinator node in the first place.

1. **List the limitations of a kernel function in CUDA.**

A CUDA program consists of code that are executed in either CPU or GPU. The code that runs on the GPU consists of called kernel function which might have several limitations compared to regular programs that we write for CPU.

Some of those limitations of kernel function in CUDA are:

1. **No recursion since there’s no call stack:** Since recursion will require call stack, we cannot perform any kind of recursion and anything involving call stack.
2. **No static variables:** This means there will be no state saved between different function invocations.
3. **Function cannot have varying number of arguments**
4. **Cannot access the host memory**
5. **Must have void return type:** This means only way to share the memory would be by passing the pointer to function parameter.
6. **Compare the differences between data parallelism and model parallelism.**

In data parallelism, data that is used to train a model is distributed between various workers, that is, some parts of data will be fed into one worker while the others will be fed into different workers. Then each of the worker will try to estimate same parameters, and they might exchange their estimates in either synchronous or asynchronous fashion. In other word, here models get replicated in each of the worker nodes. Each replica needs to then get updated about gradient calculated by the other workers. There are two ways the communication between different replicas takes place: synchronous training and asynchronous training.

However, in model parallelism, the whole data is sent to all the workers and each of the worker will independently try to estimate different parameters. After each of the parameters are estimated, nodes might exchange their estimates. In other word, the neural network itself gets split across multiple devices. This is used mainly in cases where the models consist of large number of parameters which would not fit on a single device.