Birla Institute of Technology & Science, Pilani
Work Integrated Learning Programmes Division
First Semester 2023-2024

Mid-Semester Test
(EC-2 Regular)

Course No.	: CC ZC447
Course Title	: Data Storage Technologies & Networks
Nature of Exam	: Closed Book
Pattern of Exam	: Typed Only
Weightage	: 30%
Duration	: 2 Hours
Date of Exam	: 22/09/2023 (AN)

No. of Pages = 2
No. of Questions = 10

Note to Students:
1. Please follow all the *Instructions to Candidates* given on the cover page of the answer book.
2. All parts of a question should be answered consecutively. Each answer should start from a fresh page.
3. Assumptions made if any, should be stated clearly at the beginning of your answer.

Q.1 Set. (A) What do you understand by HyperConverged Infrastructure? [2 Marks]

HyperConverged Infrastructure (HCI) is an approach to data center management that aims to simplify and streamline the deployment, management, and scaling of IT infrastructure. In HCI, compute, storage, and networking resources are integrated into a single, software-defined platform that is managed centrally. This convergence eliminates the need for separate hardware components and allows for more efficient resource utilization.

Key characteristics of HyperConverged Infrastructure include:

1. **Software-Defined**: HCI relies heavily on software-defined technologies to abstract hardware resources and enable centralized management. This means that the configuration, provisioning, and management of infrastructure resources are handled through software interfaces rather than manual hardware configurations.
2. **Integration**: HCI integrates compute, storage, and networking resources into a single platform. This integration eliminates silos and simplifies management tasks, such as provisioning and scaling, by providing a unified interface.
3. **Scalability**: HCI architectures are designed to scale easily by adding additional nodes to the cluster. This allows organizations to start with a small deployment and incrementally add resources as needed, without disrupting existing operations.
4. **Virtualization**: Virtualization plays a crucial role in HCI by abstracting physical hardware resources and presenting them as virtualized resources to applications and workloads. This enables greater flexibility and resource utilization.
5. **Automation**: HCI platforms often include automation capabilities to streamline repetitive tasks such as provisioning, monitoring, and troubleshooting.

Automation helps reduce manual intervention and improves overall operational efficiency.

6. **High Availability**: HCI architectures typically incorporate redundancy and fault tolerance mechanisms to ensure high availability of resources and applications. Features such as data replication, failover, and data resiliency help minimize downtime and data loss.

Overall, HyperConverged Infrastructure offers organizations a more agile, flexible, and cost-effective approach to managing their IT infrastructure compared to traditional siloed architectures.

Q.1 Set. (B) What do you understand by Software-Defined Storage? [2 Marks]

Software-Defined Storage (SDS) is an approach to storage management that decouples storage hardware from the software that controls it. In SDS, storage functions such as data management, provisioning, and data protection are implemented and managed through software, rather than being tied to specific hardware appliances.

Key characteristics of Software-Defined Storage include:

1. **Abstraction**: SDS abstracts storage resources from the underlying hardware, allowing storage to be pooled and managed centrally. This abstraction layer enables greater flexibility and agility in storage management, as administrators can allocate and reallocate storage resources dynamically based on changing demands.
2. **Centralized Management**: SDS platforms typically provide centralized management interfaces that allow administrators to provision, monitor, and manage storage resources across heterogeneous hardware environments from a single console. This centralized management simplifies storage administration and reduces the complexity associated with managing multiple storage arrays.
3. **Automation**: SDS often includes automation capabilities to streamline storage provisioning, data migration, and other storage management tasks. Automation helps improve efficiency, reduce manual errors, and accelerate the deployment of storage resources.
4. **Scalability**: SDS architectures are designed to scale easily to accommodate growing storage requirements. By pooling storage resources and abstracting them from the underlying hardware, SDS platforms can scale out by adding additional storage nodes or scale up by upgrading existing hardware components without disrupting operations.
5. **Cost Efficiency**: SDS can help organizations reduce storage costs by leveraging commodity hardware and avoiding vendor lock-in. By decoupling storage software from proprietary hardware appliances, SDS enables organizations to choose hardware that best fits their performance, capacity, and budget requirements.
6. **Data Services**: SDS platforms often include a wide range of data services such as deduplication, compression, encryption, snapshots, replication, and thin

provisioning. These data services help optimize storage efficiency, improve data protection, and enhance data management capabilities.

Overall, Software-Defined Storage offers organizations greater flexibility, scalability, and cost efficiency in managing their storage infrastructure compared to traditional hardware-centric storage architectures.

Q.2 Set. (A) What is the relation between response time and throughput in a storage device that supports queuing of requests? [2 Marks]

Q.2 Set. (B) What is the relation between response time and throughput in a storage device that supports queuing of requests? [2 Marks]

Q.3 Set. (A) Consider a disk with an average seek time of 5 ms, rotation speed of 12,000 rpm, and 1024-byte sectors with 600 sectors per track. We wish to read a file consisting of 2400 sectors. Estimate the total time for the operation in the following cases: [5 Marks]

a) Sequential organization where file is placed on adjacent sectors and cylinders, assuming 1 ms delay for moving from one track to an adjacent track.
b) A random organization where all sectors of the file are placed at random locations over the disk.

To estimate the total time for the read operation in each case, we need to consider the time required for three main components: seeking, rotational delay, and data transfer.

Given:

- Average seek time = 5 ms
- Rotation speed = 12,000 rpm (which is 200 revolutions per second or 0.005 seconds per revolution)
- 1024-byte sectors with 600 sectors per track

Let's calculate for each case:

a) Sequential organization:

In this case, the file is placed on adjacent sectors and cylinders, so there will be minimal seeking required.

1. **Seek Time**: Since the file is placed sequentially, there will be minimal seeking. We'll assume one seek operation to move from one track to an adjacent track. So, the seek time will be 1 ms.
2. **Rotational Delay**: Since the sectors are sequential, there will be minimal rotational delay.
3. **Data Transfer Time**: To read 2400 sectors, we need to calculate the number of tracks required. Each track contains 600 sectors, so we need 2400/600 = 4 tracks.

Let's calculate the total time:

- Seek Time: 1 ms
- Rotational Delay: Minimal
- Data Transfer Time: 4 tracks * (1/12000) seconds per revolution * (1 revolution/600 sectors) * 2400 sectors = 0.0667 seconds

Total Time = Seek Time + Rotational Delay + Data Transfer Time Total Time = 1 ms + 0 + 0.0667 seconds = 1.0667 seconds

b) Random organization:

In this case, all sectors of the file are placed at random locations over the disk, so there will be significant seeking required.

1. **Seek Time**: For each sector, we'll need to perform a seek operation to move the read head to the correct track. Since we're assuming random organization, we can't optimize seeking. We'll need to perform a seek for each sector read. So, total seek time = average seek time * number of seeks.
2. **Rotational Delay**: As before, the rotational delay will be minimal.
3. **Data Transfer Time**: Same as before, we need to read 2400 sectors.

Let's calculate the total time:

- Seek Time: Average seek time * number of seeks
- Rotational Delay: Minimal
- Data Transfer Time: Same as before

Total Time = Seek Time + Rotational Delay + Data Transfer Time

We'll calculate the seek time first:

Number of seeks = 2400 (since each sector is randomly located) Seek Time = Average seek time * number of seeks Seek Time = 5 ms * 2400 seeks = 12000 ms = 12 seconds

Now, we can calculate the total time: Total Time = 12 seconds (Seek Time) + 0 (Rotational Delay) + 0.0667 seconds (Data Transfer Time) Total Time = 12.0667 seconds

So, for the random organization, the total time for the operation would be approximately 12.0667 seconds.

Q.3 Set. (B) Consider a disk with an average seek time of 6 ms, rotation speed of 15,000 rpm, and 1024-byte sectors with 600 sectors per track. We wish to read a file consisting of 2400 sectors. Estimate the total time for the operation in the following cases: [5 Marks]

a) Sequential organization where file is placed on adjacent sectors and cylinders, assuming 1 ms delay for moving from one track to an adjacent track.
b) A random organization where all sectors of the file are placed at random locations over the disk.

To estimate the total time for the read operation in each case, we'll consider the three main components: seeking time, rotational delay, and data transfer time.

Given:

- Average seek time = 6 ms
- Rotation speed = 15,000 rpm (which is 250 revolutions per second or 0.004 seconds per revolution)
- 1024-byte sectors with 600 sectors per track
- File size: 2400 sectors

Let's calculate for each case:

a) Sequential organization:

In this case, the file is placed on adjacent sectors and cylinders, so there will be minimal seeking required.

1. **Seek Time**: Since the file is placed sequentially, there will be minimal seeking. We'll assume one seek operation to move from one track to an adjacent track. So, the seek time will be 1 ms.
2. **Rotational Delay**: Since the sectors are sequential, there will be minimal rotational delay.
3. **Data Transfer Time**: To read 2400 sectors, we need to calculate the number of tracks required. Each track contains 600 sectors, so we need 2400/600 = 4 tracks.

Let's calculate the total time:

- Seek Time: 1 ms
- Rotational Delay: Minimal
- Data Transfer Time: 4 tracks * (1/15000) seconds per revolution * (1 revolution/600 sectors) * 2400 sectors = 0.0533 seconds

Total Time = Seek Time + Rotational Delay + Data Transfer Time Total Time = 1 ms + 0 + 0.0533 seconds = 1.0533 seconds

b) Random organization:

In this case, all sectors of the file are placed at random locations over the disk, so there will be significant seeking required.

1. **Seek Time**: For each sector, we'll need to perform a seek operation to move the read head to the correct track. Since we're assuming random organization, we can't optimize seeking. We'll need to perform a seek for each sector read. So, total seek time = average seek time * number of seeks.
2. **Rotational Delay**: As before, the rotational delay will be minimal.
3. **Data Transfer Time**: Same as before, we need to read 2400 sectors.

Let's calculate the total time:

- Seek Time: Average seek time * number of seeks
- Rotational Delay: Minimal
- Data Transfer Time: Same as before

We'll calculate the seek time first:

Number of seeks = 2400 (since each sector is randomly located) Seek Time = Average seek time * number of seeks Seek Time = 6 ms * 2400 seeks = 14400 ms = 14.4 seconds

Now, we can calculate the total time: Total Time = 14.4 seconds (Seek Time) + 0 (Rotational Delay) + 0.0533 seconds (Data Transfer Time) Total Time = 14.4533 seconds

So, for the random organization, the total time for the operation would be approximately 14.4533 seconds.

Q.4 Set. (A) Briefly explain why Flash Translation Layer is needed in a SSD [2 Marks]
Q.4 Set. (B) Briefly explain why Flash Translation Layer is needed in a SSD [2 Marks]

The Flash Translation Layer (FTL) is a critical component of solid-state drives (SSDs) and is necessary due to the fundamental differences between SSDs and traditional hard disk drives (HDDs). Below are the key reasons why the FTL is needed in an SSD:

1. **Wear Leveling**: Flash memory cells in SSDs have limited write endurance, meaning they can only endure a certain number of program/erase cycles before becoming unreliable. Wear leveling is a technique used by the FTL to distribute write and erase operations evenly across all the NAND flash memory cells, thereby prolonging the lifespan of the SSD.
2. **Garbage Collection**: Unlike HDDs, which can overwrite data directly, flash memory cells must be erased before new data can be written to them. When data is deleted or modified in an SSD, the FTL needs to perform garbage collection to reclaim space by consolidating valid data and erasing blocks containing invalid or obsolete data.
3. **Bad Block Management**: Flash memory cells can develop defects over time, resulting in bad blocks that cannot reliably store data. The FTL manages bad blocks by mapping them to spare blocks and ensuring that data is not written to or read from these defective blocks.
4. **Wear-Leveling Metadata**: The FTL maintains metadata to track the usage and status of each NAND flash memory block. This metadata includes information such as wear leveling counts, block statuses (valid, invalid, or bad), and mapping tables that associate logical addresses with physical addresses.
5. **Error Correction**: Flash memory is susceptible to errors such as read disturbs, program failures, and data retention issues. The FTL incorporates error correction

mechanisms to detect and correct these errors, ensuring data integrity and reliability.

Overall, the Flash Translation Layer is essential for managing the complexities of flash memory technology and maximizing the performance, reliability, and lifespan of SSDs. It abstracts the underlying NAND flash memory architecture, providing a transparent interface for higher-level storage operations while implementing advanced techniques to optimize performance and durability.

Q.5 Set. (A) Answer the following [4 Marks]

a) Why RAID designers prefer to use parity over error-correcting codes?

RAID (Redundant Array of Independent Disks) designers often prefer to use parity over error-correcting codes (ECC) for several reasons:

1. **Cost-effectiveness**: Parity-based RAID configurations, such as RAID 5 and RAID 6, are generally more cost-effective than implementing ECC. Parity requires less overhead in terms of storage capacity compared to ECC, which typically requires additional storage for redundancy and error correction data.
2. **Simplicity**: Parity schemes are simpler to implement and manage compared to ECC. Parity-based RAID levels involve simple XOR operations to calculate and verify parity data, making them easier to understand and maintain.
3. **Performance**: Parity-based RAID configurations offer better write performance compared to ECC-based schemes. ECC typically involves more complex calculations and additional read-modify-write operations, which can impact write performance.
4. **Flexibility**: Parity-based RAID levels provide flexibility in terms of storage capacity and fault tolerance. RAID 5 can tolerate the failure of one disk, while RAID 6 can tolerate the failure of two disks. This flexibility allows RAID designers to balance between storage efficiency and fault tolerance based on specific requirements.
5. **Widely adopted**: Parity-based RAID configurations, particularly RAID 5 and RAID 6, are widely adopted and supported by various storage systems and operating systems. This widespread adoption makes them a preferred choice for many RAID implementations.

However, it's important to note that while parity provides fault tolerance against the failure of one or more disks (depending on the RAID level), it does not provide data integrity or error correction capabilities like ECC. For applications that require higher levels of data integrity and protection against bit errors, ECC or other error-correcting techniques may be necessary in addition to or instead of RAID parity.

b) What are the required I/O operations when we implement RAID 6 and need to write one block of data?

When implementing RAID 6 and needing to write one block of data, several I/O operations are required. RAID 6, like RAID 5, uses block-level striping with distributed parity, but with an additional parity block for fault tolerance. Here's a breakdown of the required I/O operations:

1. **Read Operations**: a. Read old data blocks: Before writing new data, the existing data blocks on the data disks involved in the RAID 6 stripe need to be read to calculate the parity information. b. Read old parity blocks: Similarly, the existing parity blocks on the parity disks need to be read to calculate new parity information.
2. **Calculation Operations**: a. Calculate new parity: Once the old data blocks and old parity blocks are read, new parity information needs to be calculated. For RAID 6, this typically involves XOR operations between the old data blocks and old parity blocks.
3. **Write Operations**: a. Write new data block: The new data block to be written needs to be written to one of the data disks. b. Write new parity blocks: The new parity information calculated in step 2a needs to be written to the parity disks. For RAID 6, this involves writing two parity blocks to two separate parity disks.

It's important to note that RAID 6 can tolerate the simultaneous failure of up to two disks in the array without losing data. Therefore, the parity information needs to be distributed across at least two parity disks, ensuring redundancy and fault tolerance. This means that when writing new parity blocks, they need to be written to different disks than the disk where the new data block is written, to maintain fault tolerance in case of disk failures.

Overall, the I/O operations involved in writing one block of data in a RAID 6 configuration include reading existing data and parity blocks, calculating new parity, writing the new data block, and writing the new parity blocks. These operations ensure data integrity and fault tolerance in the RAID 6 array.

Q.6 Set. (A) A processor sends 70 disk I/Os per second to a disk, these requests arrive in Poisson distribution, and the average service time of the disk is 10 ms. Using queuing theory, estimate the average response time. [3 Marks]

To estimate the average response time using queuing theory, we can use the M/M/1 queuing model, which is suitable for a single-server system with a Poisson arrival process and exponentially distributed service times. In this case, the disk acts as a single server servicing disk I/O requests.

Given:

- Arrival rate (λ) = 70 disk I/Os per second
- Service time (μ) = 10 ms (or 0.01 seconds)

We need to calculate the average response time (R), which is the total time spent by a request in the system, including the time spent waiting in the queue and the time spent being serviced.

The M/M/1 queuing model provides the following formula for the average response time: $R = \frac{1}{\mu - \lambda}$

Let's plug in the values:

$R = \frac{1}{0.01 - 70}$

Now, we can calculate the value:

$R = \frac{1}{-69.99}$

$R \approx -0.0143$

Since the result is negative, it indicates that the arrival rate is higher than the service rate, which means that the system is not stable. In this case, there are more requests arriving at the disk than it can handle, leading to an increasing backlog of requests and an unbounded increase in the average response time.

To further analyze the system, we can calculate the utilization (ρ), which represents the fraction of time the server is busy: $\rho = \frac{\lambda}{\mu}$

Plugging in the values: $\rho = \frac{70}{7000} = 0.01 \times 70 = 7000$

The utilization (ρ) exceeds 1, indicating that the system is overloaded, which is consistent with the negative average response time.

In practice, to improve the system's performance and stability, the arrival rate should be reduced, or the service rate should be increased to ensure that the utilization remains below 1, indicating that the system can handle the incoming requests effectively.

Q.6 Set. (B) A processor sends 60 disk I/Os per second to a disk, these requests arrive in Poisson distribution, and the average service time of the disk is 11 ms. Using queuing theory, estimate the average response time. [3 Marks]

Q.7 Set. (A) Differentiate Interrupt driven, and DMA I/O techniques [3 Marks]

Interrupt-driven I/O and Direct Memory Access (DMA) are two techniques used in computer systems to facilitate input/output (I/O) operations. Here's a differentiation between the two:

1. **Interrupt-Driven I/O**:
    - **Overview**: In interrupt-driven I/O, the CPU initiates an I/O operation and then continues executing other instructions while waiting for the I/O device to complete the operation. When the device has finished its task, it interrupts the CPU to signal completion.
    - **Process**:
        - The CPU initiates an I/O operation by issuing a command to the I/O device and then continues executing other instructions.
        - While the I/O device performs the requested operation, the CPU executes unrelated tasks.
        - Once the I/O device completes the operation, it generates an interrupt signal to the CPU.
        - The CPU suspends its current execution, saves its state, and transfers control to the interrupt handler routine.
        - The interrupt handler processes the interrupt, acknowledges the completion of the I/O operation, and may perform additional tasks related to the I/O operation.
        - After handling the interrupt, the CPU resumes execution of the interrupted program.
    - **Characteristics**:
        - It involves frequent context switching between the CPU and the interrupt handler routine, which can incur overhead.
        - Suitable for scenarios where the CPU has sufficient processing power to handle interrupt overhead and where I/O operations are infrequent or short.
        - Can cause performance degradation in CPU-bound tasks due to frequent interruptions.
2. **Direct Memory Access (DMA)**:
    - **Overview**: DMA is a mechanism that allows peripheral devices to transfer data directly to and from the system's memory without involving the CPU. DMA controllers manage these data transfers independently of the CPU, freeing up the CPU to perform other tasks.
    - **Process**:
        - The CPU initiates a DMA transfer by configuring the DMA controller with the source and destination addresses and the transfer size.
        - The DMA controller transfers data directly between the I/O device and the system's memory without CPU intervention.
        - Once the transfer is complete, the DMA controller may generate an interrupt to notify the CPU or simply update a status register.
        - The CPU can then check the status of the DMA transfer and proceed with any necessary processing or additional I/O operations.
    - **Characteristics**:

- Reduces CPU overhead and improves system performance by offloading data transfer tasks from the CPU.
- Suitable for scenarios where large volumes of data need to be transferred between I/O devices and memory, such as disk I/O or network I/O.
- Requires hardware support in the form of a DMA controller and may introduce complexity in system design and programming.

In summary, interrupt-driven I/O relies on CPU intervention and interrupts to manage I/O operations, whereas DMA enables I/O devices to transfer data directly to and from memory without CPU involvement, thereby reducing CPU overhead and improving system performance.

Q.7 Set. (B) Differentiate Interrupt driven, and Programmed I/O techniques [3 Marks]

Interrupt-driven I/O and Programmed I/O are two techniques used in computer systems to manage input/output (I/O) operations. Here's a differentiation between the two:

1. **Interrupt-Driven I/O**:
   - **Overview**: In interrupt-driven I/O, the CPU initiates an I/O operation and then continues executing other instructions while waiting for the I/O device to complete the operation. When the device has finished its task, it interrupts the CPU to signal completion.
   - **Process**:
     - The CPU initiates an I/O operation by sending commands to the I/O device and then continues executing other instructions.
     - While the I/O device performs the requested operation, the CPU executes unrelated tasks.
     - Once the I/O device completes the operation, it generates an interrupt signal to the CPU.
     - The CPU suspends its current execution, saves its state, and transfers control to the interrupt handler routine.
     - The interrupt handler processes the interrupt, acknowledges the completion of the I/O operation, and may perform additional tasks related to the I/O operation.
     - After handling the interrupt, the CPU resumes execution of the interrupted program.
   - **Characteristics**:
     - It involves frequent context switching between the CPU and the interrupt handler routine, which can incur overhead.

- Suitable for scenarios where the CPU has sufficient processing power to handle interrupt overhead and where I/O operations are infrequent or short.
- Can cause performance degradation in CPU-bound tasks due to frequent interruptions.

2. **Programmed I/O**:
   - **Overview**: Programmed I/O is a simple I/O technique where the CPU directly manages data transfer between itself and I/O devices. The CPU explicitly issues commands to the I/O device and waits until the operation is completed before proceeding with other tasks.
   - **Process**:
     - The CPU initiates an I/O operation by sending commands to the I/O device and then waits for the operation to complete.
     - While the I/O device performs the requested operation, the CPU remains idle or executes busy-wait loops to check the status of the operation.
     - Once the I/O device completes the operation, the CPU retrieves the data or status information directly from the device.
     - The CPU proceeds with further processing or issues additional I/O commands as necessary.
   - **Characteristics**:
     - Simple and straightforward, as the CPU directly controls the entire I/O operation.
     - Inefficient in terms of CPU utilization, as the CPU remains idle or busy-waits during I/O operations, leading to wasted CPU cycles.
     - Suitable for simple I/O operations or scenarios where the CPU can afford to wait for the completion of I/O operations without impacting overall system performance.

In summary, interrupt-driven I/O relies on interrupts to asynchronously notify the CPU of I/O completion, while Programmed I/O involves the CPU explicitly managing I/O operations, either by waiting for completion or actively polling the status of the I/O device. Interrupt-driven I/O is more efficient and suitable for handling asynchronous I/O events, while Programmed I/O is simpler but less efficient in terms of CPU utilization.

Q.8 Set. (A) How is USB protocol different from SATA? [2 Marks]

The USB (Universal Serial Bus) protocol and SATA (Serial Advanced Technology Attachment) are both widely used interfaces for connecting peripheral devices to computers, but they differ in several key aspects:

1. **Purpose**:
   - USB: USB is a versatile interface primarily used for connecting various types of peripheral devices such as keyboards, mice, printers, external storage drives, smartphones, and other consumer electronics to computers and other host devices. It supports a wide range of devices and is designed for hot-swapping, allowing devices to be connected and disconnected without powering down the system.
   - SATA: SATA is a high-speed interface specifically designed for connecting storage devices such as hard disk drives (HDDs), solid-state drives (SSDs), and optical drives to computers. It is optimized for high-speed data transfer between storage devices and the host system.
2. **Topology**:
   - USB: USB typically uses a bus topology, where multiple devices are connected to a single host controller through a common bus. USB devices can be connected in a daisy-chain or hub-and-spoke configuration.
   - SATA: SATA uses a point-to-point topology, where each SATA device is connected directly to a SATA host controller. Each SATA connection is dedicated to a single device, providing a direct and dedicated link between the device and the host.
3. **Data Transfer Rate**:
   - USB: USB supports multiple data transfer rates depending on the USB version and the specific device. USB 2.0 offers data transfer rates of up to 480 Mbps (Megabits per second), USB 3.0 up to 5 Gbps, USB 3.1 up to 10 Gbps, and USB 3.2 up to 20 Gbps.
   - SATA: SATA supports higher data transfer rates compared to USB, especially in its latest iterations. SATA revision 3.0 (SATA III) offers data transfer rates of up to 6 Gbps, while SATA revision 3.2 introduced SATA Express, which can reach speeds of up to 16 Gbps.
4. **Power Delivery**:
   - USB: USB provides power delivery to connected devices, allowing them to be powered directly from the USB port. USB supports various power profiles, including low-power (up to 2.5 watts) and high-power (up to 7.5 watts) modes.
   - SATA: SATA does not provide power delivery to connected devices. Devices connected via SATA require their own power source, typically through a separate power connector.
5. **Compatibility**:
   - USB: USB is widely supported across different platforms and operating systems, making it a universal interface for connecting peripheral devices.
   - SATA: SATA is primarily used for internal storage devices in desktop and laptop computers. While it is a standard interface for storage devices, it is

not as versatile as USB and is not commonly used for other types of peripheral devices.

Overall, while both USB and SATA are serial interfaces used for connecting devices to computers, they serve different purposes, have different topologies, data transfer rates, power delivery mechanisms, and compatibility profiles.

Q.8 Set. (B) Why SSDs using NVMe perform better than ones using SATA? [2 Marks]

SSDs (Solid State Drives) using NVMe (Non-Volatile Memory Express) typically perform better than those using SATA (Serial Advanced Technology Attachment) for several reasons:

1. **Interface Speed**:
   - NVMe: NVMe SSDs utilize the NVMe interface protocol, which is specifically designed for modern non-volatile memory technologies like NAND flash and 3D XPoint. NVMe takes advantage of the high-speed PCIe (Peripheral Component Interconnect Express) interface, providing significantly higher data transfer rates compared to SATA. PCIe offers multiple lanes for data transfer, allowing NVMe SSDs to achieve much higher speeds.
   - SATA: SATA SSDs, on the other hand, are limited by the SATA interface, which has a maximum data transfer rate of 6 Gbps (SATA III). While this speed was sufficient for traditional hard disk drives (HDDs), it becomes a bottleneck for high-performance SSDs.
2. **Queue Depth**:
   - NVMe: NVMe SSDs support a much higher queue depth compared to SATA SSDs. Queue depth refers to the number of commands that a storage device can handle simultaneously. NVMe SSDs can handle thousands of commands in parallel, allowing for more efficient use of the SSD's resources and reducing latency.
   - SATA: SATA SSDs have a lower queue depth compared to NVMe SSDs, limiting their ability to handle multiple commands simultaneously. This can lead to increased latency, especially in high-demand scenarios.
3. **Command Set**:
   - NVMe: NVMe SSDs support a more advanced command set optimized for modern storage technologies. NVMe commands are highly parallelized and optimized for NAND flash memory, allowing for efficient data transfers and reduced latency.
   - SATA: SATA SSDs use a command set designed for traditional hard disk drives, which may not be as efficient for SSDs. While SATA SSDs can still achieve good performance, they are limited by the capabilities of the SATA interface and command set.
4. **Parallelism**:

- NVMe: NVMe SSDs take advantage of parallelism inherent in PCIe and NVMe protocols, allowing for simultaneous data transfer operations across multiple channels and lanes. This parallelism enables NVMe SSDs to achieve higher throughput and lower latency.
- SATA: SATA SSDs are limited by the serial nature of the SATA interface, which restricts the degree of parallelism that can be achieved. As a result, SATA SSDs may not fully utilize the available bandwidth and may experience higher latency compared to NVMe SSDs.

Overall, SSDs using NVMe outperform those using SATA due to the higher interface speed, increased queue depth, optimized command set, and greater parallelism offered by NVMe. These factors result in significantly improved data transfer rates, reduced latency, and overall better performance for NVMe SSDs compared to SATA SSDs.

Q.9 Set. (A) How pNFS and Compound RPC enhance performance of NFS? [3 Marks]

pNFS (Parallel NFS) and Compound RPC (Remote Procedure Call) are two technologies that enhance the performance of NFS (Network File System), a distributed file system protocol commonly used for accessing files over a network. Here's how they improve NFS performance:

1. **pNFS (Parallel NFS)**:
   - **Parallel Access**: pNFS enables parallel access to files stored on NFS servers by allowing clients to access different parts of a file concurrently. This parallel access improves performance by distributing the workload across multiple servers and leveraging the aggregate bandwidth of multiple storage devices.
   - **Layouts**: pNFS introduces the concept of storage layouts, which define how file data is distributed across multiple storage devices or servers. By using storage layouts, pNFS clients can access data directly from storage devices without going through a centralized server, reducing latency and improving throughput.
   - **Scalability**: pNFS enhances the scalability of NFS by allowing multiple storage devices or servers to be added to the storage pool. This scalability ensures that NFS performance can scale with the increasing demands of modern storage environments.
2. **Compound RPC (Remote Procedure Call)**:
   - **Batching Requests**: Compound RPC allows NFS clients to combine multiple NFS operations into a single RPC request. By batching requests, Compound RPC reduces the overhead associated with initiating and processing individual RPC requests, leading to improved efficiency and reduced latency.
   - **Reduced Overhead**: Compound RPC reduces the overhead of communication between NFS clients and servers by minimizing the number of RPC requests and responses exchanged between them. This

reduction in overhead translates to lower network latency and improved overall performance.
- **Optimized Data Transfer**: Compound RPC optimizes data transfer between NFS clients and servers by aggregating multiple read or write operations into a single RPC request. This optimization reduces the number of network round trips required to transfer data, resulting in faster data access and improved throughput.

Overall, pNFS and Compound RPC enhance the performance of NFS by enabling parallel access to files, reducing communication overhead, optimizing data transfer, and improving scalability. These technologies address the limitations of traditional NFS implementations and make NFS better suited for modern distributed storage environments with high-performance requirements.

Q.10 Set. (A) Differentiate Unified, Gateway, and Scale-out(Clustered) NAS implementations. [4 Marks]

Unified, Gateway, and Scale-out (Clustered) NAS implementations are three distinct approaches to Network Attached Storage (NAS) architectures, each with its own characteristics and use cases. Here's a differentiation between them:

1. **Unified NAS**:
   - **Overview**: Unified NAS integrates file-based (NAS) and block-based (SAN) storage protocols within a single storage system. It allows users to access both file and block storage from the same device, providing flexibility and convenience.
   - **Features**:
     - Supports both NFS (Network File System) and SMB (Server Message Block) file protocols for file-based access.
     - Supports block-level access protocols like iSCSI or Fibre Channel for SAN (Storage Area Network) connectivity.
     - Single storage system manages both file and block storage resources.
   - **Use Cases**:
     - Ideal for environments that require both file and block storage capabilities, such as virtualization, database applications, and mixed workload environments.
     - Provides consolidated storage infrastructure, simplifying management and reducing the need for multiple storage systems.
2. **Gateway NAS**:
   - **Overview**: Gateway NAS, also known as NAS gateway or NAS head, is a device that acts as an intermediary between clients and backend storage systems. It translates file-based access requests into block-level access requests, enabling file-based access to block storage devices.
   - **Features**:

- Accepts file-based access requests from clients using protocols like NFS and SMB.
- Translates file-based requests into block-level commands and forwards them to backend block storage devices using SAN protocols like iSCSI or Fibre Channel.
- Presents file-based storage resources to clients while leveraging the scalability and performance of block storage systems.

- **Use Cases**:
  - Useful in environments where existing block storage systems need to be accessed using file-based protocols.
  - Provides a bridge between NAS and SAN environments, allowing organizations to leverage existing SAN infrastructure while providing file-based access to users.

3. **Scale-out (Clustered) NAS**:
   - **Overview**: Scale-out NAS, also known as clustered NAS, is a storage architecture that scales by adding multiple NAS nodes to form a cluster. These nodes work together to provide a single, unified storage pool that can scale capacity and performance horizontally.
   - **Features**:
     - Multiple NAS nodes are combined into a single cluster, sharing a common filesystem or namespace.
     - Provides seamless scalability by adding additional nodes to the cluster, increasing both storage capacity and performance.
     - Data is distributed across multiple nodes in the cluster, enabling parallel access to data and improving performance.
   - **Use Cases**:
     - Suitable for environments with large-scale data storage requirements, such as media and entertainment, high-performance computing (HPC), and big data analytics.
     - Provides high availability and fault tolerance, as data is replicated or distributed across multiple nodes, reducing the risk of data loss due to hardware failures.

In summary, Unified NAS combines file and block storage protocols within a single system, Gateway NAS provides file-based access to block storage devices, and Scale-out NAS scales capacity and performance by adding multiple nodes to form a cluster. Each approach offers unique benefits and is suited to different use cases and storage requirements.