

1. Cloud Storage Fundamentals

- **(a)** Traditional file systems often struggle with horizontal scalability, metadata limitations, and distributed access, making them less suitable for modern applications.
- **(b)** Object storage treats data as discrete objects in a flat address space, each linked to metadata and accessed via unique IDs, enhancing data management.
- **(c)** Key benefits of object storage include easy scalability, flexible metadata, redundancy, HTTP access, and high durability, making it ideal for cloud environments.
- **(d)** Object storage is metadata-rich and scalable, block storage is optimized for high I/O in databases, file storage is hierarchical for traditional uses, and key-value storage offers minimal structure for fast lookups.
- **(e)** Scaling up involves enhancing a single machine, which can be costly, while scaling out means adding more similar machines, which is more economical and flexible. Better code can also optimize software efficiency and reduce complexity.
- **(f) Why Scaling Out is Less Expensive than Scaling Up:**

Scaling out (horizontal scaling) involves adding/buying more similar machines resulting in cheaper and better results, while scaling up (vertical scaling) involves buying or upgrading a single better machine.

Cost Efficiency:

- **Hardware Costs:** Commodity hardware used in scaling out is generally cheaper than high-end, upgraded hardware required for scaling up.
 - **Redundancy:** Scaling out provides built-in redundancy by utilizing multiple machines, which can reduce costs in downtime and recovery.
 - **Flexibility:** Scaling out allows for incremental increases in capacity and performance, making it easier to adapt to changing workloads.
- **(g) Design Aspects of Object Storage Enabling Scaling Out:**
 - **Decentralized Architecture:** Object storage often uses a distributed model, allowing multiple nodes to collaborate without a single point of failure.
 - **Data Distribution:** Objects are stored across various nodes which can be added or removed as needed, enabling easy scaling.
 - **Metadata Management:** Object storage systems manage data as objects with metadata, allowing efficient indexing and retrieval.

- **Cost-Effectiveness:** This architecture can utilize lower-cost hardware, making it economically viable to scale out.
- **(h) Components of a Data Centre:**
 - **Racks:** Structures that house servers and networking equipment.
 - **Server Nodes:** Individual servers that process data and run applications.
 - **Storage Nodes:** Dedicated devices for storing data, which may include hard drives or SSDs.
 - **Networking Equipment:** Routers, switches, and cables that facilitate communication between components.
 - **Power and Cooling Systems:** Infrastructure to provide necessary power and maintain optimal operating temperatures.
- **(i) Difference Between Storage, Memory, CPU, and Network in a Cluster:**
 - **Storage:** Refers to where data is physically stored (e.g., hard drives, SSDs).
 - **Memory (RAM):** Temporary storage that provides quick access to data for processes currently in use, impacting speed and performance.
 - **CPU:** The central processing unit executes instructions and performs calculations; it is crucial for processing tasks.
 - **Network:** The system of connections and protocols that facilitate communication between nodes in a cluster, affecting data transfer rates and overall system performance.
- **(j) Typical Numbers for Per-Node Specifications:**
 - **Storage Capacity:** Typically ranges from a few terabytes (TB) to petabytes (PB) per node, depending on the application.
 - **Memory:** Commonly between 64 GB to several terabytes of RAM, depending on workloads.
 - **Cores:** The number of CPU cores can range from 4 to over 64, again depending on task requirements.
- **(k) Comparison of Storage and Memory Technologies:**
 - **HDD (Hard Disk Drive):**
 - **Capacity:** High capacity, often in several TBs.
 - **Throughput:** Slower read/write speeds compared to SSDs.

➤ **SSD (Solid State Drive):**

- **Capacity:** Generally lower than HDD but rapidly increasing.
- **Throughput:** Much faster than HDD, resulting in quick data access and lower latency.

➤ **RAM (Random Access Memory):**

- **Capacity:** Typically ranges from 8 GB to several TBs in enterprise systems.
- **Throughput:** Extremely high speed, enabling rapid data access and processing at runtime.

- **(l)** The key-value model maps keys to stored objects, commonly used in systems like S3 and DynamoDB, facilitating quick data retrieval.
- **(m)** Data represents the actual content, while metadata provides descriptive information such as owner, creation date, and file type, enhancing data context.
- **(n)** Cloud systems can store billions of objects, with individual object sizes reaching up to 5 TB, as seen in S3, allowing for extensive data storage capabilities.
- **(o)** Major cloud storage vendors include Amazon S3, Microsoft Azure Blob, Google Cloud Storage, and IBM Cloud, each offering unique features and services.
- **(p)** In S3 and Azure Blob, a bucket (S3) or container (Azure) serves as an object container, with various blob types available for different use cases, such as Block Blobs for large files, Append Blobs for log data, and Page Blobs for frequent read/write operations.
- **(q)** Latency refers to the response time delay for a system to respond to a request, while availability indicates system uptime, and durability measures the probability of data loss, with high durability being crucial for data integrity.
- **(r)** Common use cases for cloud **OBJECT** storage include backups, large file storage, IoT data management, video hosting, and application logs, showcasing its versatility.
- **(s)** REST APIs are HTTP-based services that utilize methods like GET, POST, PUT, and DELETE, with common status codes including 200 (OK), 404 (Not Found), and 500 (Error), facilitating easy integration with applications.
- **(t)** The CAP Theorem highlights the trade-offs between consistency, availability, and partition tolerance in distributed systems, where maintaining strict consistency is often impractical at scale, leading to design choices that prioritize availability and partition tolerance.
- **(u)** For large-scale distributed systems, CAP (Consistency, Availability, Partition Tolerance) becomes more relevant than ACID because it's hard to guarantee strict

Continued s) A Uniform Resource Identifier (URI) is a string of characters used to uniquely identify a resource on the internet, either by location (URL), name (URN), or both.

transactional integrity across many nodes. Network partitions are inevitable, so systems must balance trade-offs between consistency and availability. ACID is ideal for centralized databases, but CAP better addresses the realities of distributed, scalable systems.

2. Distributed File Systems

- **(a)** Block storage divides data into equal-sized blocks, storing them independently, while object storage manages entire objects, each accompanied by metadata and a unique identifier.
- **(b)** The key-value model organizes data as key-object pairs in a flat structure, contrasting with traditional file systems that use a hierarchical organization with folders and file paths.
- **(c)** Object storage systems can accommodate billions of objects, whereas traditional file systems face inode limits that restrict the number of files.
- **(d)** Object storage, such as Amazon S3, supports massive object sizes of up to 5 TB, while file systems and block storage typically have smaller size constraints. 128MB/Block
- **(e)** In local file systems, block sizes are typically small (e.g., **4KB**), while distributed file systems like HDFS use much larger block sizes (e.g., **128MB**). Larger blocks reduce overhead, improve throughput, and minimize the number of disk seeks required, which is crucial for big data performance.
- **(f)** HDFS offers **scalability** to store and access massive amounts of data efficiently. Its **fault tolerance** is achieved through block replication across nodes, ensuring data reliability even if some nodes fail.
- **(g)** HDFS ensures data consistency through an **append-only model**—files are only written by appending, not by overwriting. This design supports **atomic updates**, simplifies data management, and prevents file corruption in distributed environments.
- **(h)** HDFS is optimized for **high throughput**, making it ideal for batch processing and large-scale analytics. While it has **higher latency**, this is acceptable for big data use cases where the focus is on volume and speed over immediate responsiveness.
- **(i)** HDFS is cost-effective as it runs on **commodity hardware**. It provides **fault tolerance** through data replication, ensuring high availability and durability even in the face of hardware failures.
- **(j)** In **centralized architectures** like HDFS, the NameNode acts as the central control unit, which can create bottlenecks or failures. **Decentralized (peer-to-peer) architectures** distribute responsibility, enhancing **scalability** and **resilience** by eliminating single points of failure.

- **(k)** HDFS (Hadoop Distributed File System) is designed to manage large data sets across distributed environments. The **NameNode** controls metadata, including the file system structure and mapping of files to blocks. **DataNodes** store the actual data blocks and serve read/write requests from clients. **Block replication** ensures fault tolerance by storing multiple copies of data blocks across different DataNodes.
- **(l)** Clients communicate with the **NameNode** to obtain metadata (like block locations), and with **DataNodes** to read/write the actual data blocks. The **NameNode** manages the system's namespace and coordinates operations with DataNodes. **DataNodes** report back to the NameNode periodically about the blocks they store and their status. (heartbeat)
- **(m)** The **NameNode** is a single point of failure in HDFS. To prevent system downtime, we can implement a ^{Standby} **Secondary NameNode** (for checkpoints) or use **NameNode failover mechanisms** such as High Availability (HA) configurations to provide redundancy and ensure continuous operation.
- **(n)** The **namespace** represents the hierarchical structure of directories and files. The NameNode stores this **metadata in memory** for fast access. On disk, it maintains a **namespace file** (for the persistent file system structure) and an **edit log** (which records changes for recovery after restarts).
- **(o)** The Standby NameNode maintains a live copy of the active NameNode's metadata, facilitating seamless failover and enhancing system reliability.
- **(p)** Common HDFS shell commands include **-ls** for listing files, **-mkdir** for creating directories, **-put** for uploading files, **-get** for downloading files, and **-rm** for deleting files, providing users with essential tools for data management.