Q1. Why does Hadoop prioritize *data locality* instead of distributing tasks evenly across all nodes? What are the trade-offs involved?

Answer:
Hadoop prioritizes *data locality* because transferring large datasets over the network drastically slows computation. Instead of sending terabytes of data to a node, it sends a small Map task to the node where the data already resides. This minimizes network congestion and speeds up processing.
However, data locality may cause uneven task distribution: nodes storing more data get more tasks initially, while others stay idle. Hadoop mitigates this using *speculative execution* and *dynamic load balancing* — reassigning tasks as others complete. The trade-off is between *throughput* (favoring locality) and *parallel fairness* (favoring balance).

Q2. Explain in detail how **block size** in HDFS impacts data processing speed, load balancing, and fault tolerance.

Answer:
HDFS uses large block sizes (default 128 MB) to reduce seek overhead and metadata load. A larger block allows longer sequential reads, maximizing disk transfer rates and minimizing the time spent seeking between blocks. It also decreases the burden on the NameNode, which stores block metadata in RAM.
However, smaller blocks increase parallelism because more Map tasks can process data simultaneously. In practice, Hadoop chooses a balance — large enough for efficiency, small enough for good load distribution. For fault tolerance, each block is replicated (typically 3x), ensuring recovery even if nodes fail.

Q3. Compare **containerization** and **virtualization** in terms of architecture, overhead, and use in Big Data clusters.

Answer:
Virtualization emulates full hardware, running multiple guest OSes on one host via a hypervisor (Type 1 or Type 2). Each VM includes its own kernel and libraries, leading to heavy memory and CPU overhead.
Containerization, however, virtualizes at the OS level: containers share the host kernel but isolate processes using namespaces and cgroups. They start in seconds and consume minimal resources.
In Big Data clusters, containers (e.g., via Docker + Kubernetes) deploy distributed jobs quickly, isolate users securely, and scale horizontally — whereas VMs are used for full multi-OS environments or legacy setups.

Q4. Describe the sequence of operations when a Hadoop job is submitted — from the moment the client runs it until the final output is written to HDFS.

Answer:

1. **Client Submission:** The user submits a job to YARN, packaged as a JAR with Map and Reduce code, input, and output paths.

2. **ApplicationMaster Launch:** The ResourceManager allocates a container for the ApplicationMaster (AM).

3. **Task Assignment:** AM requests containers from NodeManagers for Map and Reduce tasks based on available resources and data locality.

4. **Map Execution:** Each Map task processes one input split, outputs intermediate key-value pairs.

5. **Shuffle and Sort:** Intermediate data is transferred to reducers — grouped by key, merged, and sorted.

6. **Reduce Execution:** Reducers aggregate results and write the final output to HDFS via FileOutputFormat.

7. **Job Completion:** AM reports success/failure to the client, and YARN frees resources.

Q5. What problem does **the Combiner function** solve in MapReduce? Why can't it always be applied?

Answer:
The Combiner function reduces network load by performing local aggregation of map outputs before transferring them to reducers — for example, summing counts of the same word from a mapper's output.
However, combiners can only be used when the function is both *commutative* and *associative* (e.g., sum, max). If the function's logic depends on the order or frequency of data (like computing averages or medians), the Combiner could alter final results — hence it's optional and not guaranteed by Hadoop.

Q6. In Bash scripting, explain how `IFS`, `set -e`, and `$?` interact during script execution and debugging.

Answer:
`IFS` (Internal Field Separator) determines how the shell splits input text (by spaces, tabs, or newlines). Changing it allows parsing CSVs or logs line by line.
`set -e` ensures the script exits immediately on command failure, preventing cascading errors — essential for automation pipelines.
`$?` stores the exit code of the last command (0 for success, nonzero for error). Used in debugging or conditional handling like:

```
cp data.txt /backup/
if [ $? -ne 0 ]; then echo "Copy failed"; fi
```

Together, they provide control, error handling, and safety in automated Bash workflows.

Q7. Why can't Hadoop effectively handle numerous small files, and what architectural limitation causes this?

Answer:
Each file in HDFS stores metadata (location, replication, permissions) in the **NameNode's memory**. One block's metadata requires ~150 bytes, so millions of tiny files quickly exhaust RAM.
HDFS is optimized for large, sequential reads — small files prevent efficient streaming, increase seek operations, and overload NameNode memory.
Solutions include combining small files using **HAR (Hadoop Archives)**, **SequenceFiles**, or external tools that merge and compress data before upload.

Q8. Explain how **Intel Optane** bridges the gap between RAM and SSDs. Why is it valuable for real-time analytics?

Answer:
Intel Optane (3D XPoint) acts as persistent memory with latency (~300 ns) between DRAM (~100 ns) and SSD (~100 µs). It can function as extended RAM or a high-speed storage tier.
In Big Data analytics, Optane allows caching "hot" data near memory speed without DRAM's volatility or cost. Frameworks like Spark or Flink can maintain larger in-memory datasets, reducing recomputation and improving checkpoint recovery time.

Q9. How does the Linux concept of *namespaces* and *control groups (cgroups)* form the foundation of Docker's architecture?

Answer:
Namespaces isolate system resources — PID, NET, MNT, UTS, IPC, USER — giving each container its own process tree, network stack, and filesystem view.
Cgroups enforce resource limits and priorities (CPU, memory, I/O).
Docker combines both: namespaces isolate containers; cgroups ensure fair resource sharing. This kernel-level isolation enables containers to act like lightweight virtual machines without running separate OS instances.

Q10. Why is replication across *racks* preferred in HDFS rather than keeping all replicas on the same node or rack?

Answer:
Rack-level replication prevents data loss if an entire rack (or its switch) fails. By default, Hadoop stores:

- one replica locally (node-local),
- one on a different node within the same rack,
- and one on another rack.
  This setup ensures both performance (local reads) and reliability (fault isolation). If a rack fails, the other racks still hold complete data copies, maintaining cluster availability.

Q11. Discuss how *YARN's architecture* improves upon the older Hadoop MapReduce v1 framework.

Answer:
In MRv1, the JobTracker handled both resource management and job scheduling — causing scalability bottlenecks.
YARN decouples these roles:

- **ResourceManager (RM):** cluster-wide resource allocation.
- **ApplicationMaster (AM):** per-job scheduler and monitor.
- **NodeManager:** node-level executor and reporter.
  This separation allows multiple frameworks (Spark, Tez, Flink) to share the same cluster, improves fault isolation, and scales to thousands of concurrent applications.

Q12. A Docker container exits immediately after running. Explain three possible causes and how to fix each.

Answer:

1. **No foreground process:** Containers terminate when the main process (e.g., `CMD`) ends. Fix: run a persistent command like `tail -f /dev/null` or proper entrypoint service.
2. **Incorrect ENTRYPOINT/CMD:** A malformed instruction causes premature exit. Fix: verify Dockerfile syntax (`ENTRYPOINT ["python","app.py"]`).
3. **Application crash:** Missing dependencies or permission errors crash the process. Fix: inspect logs using `docker logs <container>` and rebuild the image with required libraries.

Q13. Why are Bash scripts used for automating container and Hadoop operations instead of Python or Java?

Answer:
Bash interacts directly with the Linux shell and system binaries, making it ideal for automating infrastructure-level tasks like container creation, log cleanup, and cron jobs. It executes native commands (`docker`, `hdfs dfs`, `hadoop jar`) without compiling or importing libraries.
While Python or Java excel in logic-heavy workflows, Bash provides speed and simplicity for command orchestration — crucial in DevOps and cluster administration scripts.

Q14. What would happen if all map tasks finish but only one reducer is extremely slow (a "straggler")? How does Hadoop handle it?

Answer:
A straggler reducer delays the entire job since Hadoop can't finish until all reducers complete. YARN's scheduler detects slow tasks and launches **speculative duplicates** of the lagging task on idle nodes.
Whichever copy finishes first is accepted, and the other is terminated. This speculative execution minimizes total job time caused by hardware inconsistency or network lag.

Q15. Explain why Docker uses *layers* in images and how they improve performance and storage efficiency.

Answer:
Docker images are built in layers — each command (`FROM`, `RUN`, `COPY`) creates a new immutable layer. These layers are cached and shared among containers, so building or updating an image reuses existing parts.
For example, two images with the same base (Ubuntu) share that layer, saving disk space. Layering also speeds up rebuilds: unchanged layers are skipped, allowing incremental updates and quick deployments.

Q16. Compare **HDFS replication** with **RAID mirroring** in terms of failure scope and data recovery.

Answer:
RAID mirrors data across disks within one server — protecting against single-disk failures but not server loss. HDFS replication spreads data blocks across multiple nodes (and racks).
If one node fails, other nodes' replicas are used automatically, and Hadoop regenerates the missing copy. Thus, HDFS provides *cluster-level* fault tolerance, whereas RAID provides *hardware-level* redundancy.

Q17. A Bash script uses a while loop to read a file but outputs nothing. What could be wrong? Explain using IFS and redirection concepts.

Answer:
If written as

```
while read line; do echo "$line"; done < file.txt
```

it works correctly.
But if using `cat file.txt | while read line`, the loop runs in a subshell — variables modified inside aren't visible outside. Also, not setting `IFS` properly may trim spaces.
To fix: use `IFS= read -r line` and input redirection (`done < file.txt`) instead of pipes to preserve formatting and whitespace.

Q18. Why is "schema-on-read" used in Hadoop rather than "schema-on-write" like in relational databases?

Answer:
In RDBMS, *schema-on-write* enforces structure before storing data — fast reads but slow ingestion. Hadoop uses *schema-on-read*: raw data (structured or not) is stored as-is, and

structure is applied only when read.
This allows flexible ingestion of diverse data (CSV, JSON, logs) without predefined schemas, supporting exploratory analytics and scalability for unstructured data.

Q19. Explain how HDFS achieves *fault tolerance* and *consistency* despite using unreliable commodity hardware.

Answer:
HDFS maintains fault tolerance through **replication** and **heartbeat monitoring**. Each block is replicated (default factor 3) on separate nodes/racks.
If a DataNode fails to send heartbeats, the NameNode marks it dead and replicates its blocks elsewhere.
Consistency is maintained through a write-once, read-many model: files can't be modified once written, ensuring synchronization and preventing conflicts.

Q20. Discuss why Docker containers are preferred for deploying Hadoop or Spark clusters in modern environments.

Answer:
Containers offer isolated, reproducible environments that eliminate dependency issues across cluster nodes. Each container runs the exact Hadoop/Spark version with consistent libraries, simplifying scaling and upgrades.
They start faster than VMs, making them ideal for ephemeral clusters in CI/CD or cloud. Using Kubernetes or Docker Compose, containers can simulate multi-node Hadoop setups for labs, testing, or production with minimal overhead.

Q21. Why are large sequential read/write operations more efficient than random access in Big Data systems?

Answer:
Disk drives and even SSDs perform best with sequential operations because data is fetched continuously without frequent seeks. Random access requires repositioning the read head (HDD) or separate I/O calls (SSD), increasing latency.
Big Data frameworks like Hadoop optimize for streaming large blocks (128 MB+) sequentially, minimizing seeks and maximizing throughput for batch analytics.

Q22. What happens when you execute `docker run -d --name namenode --net hadoop-net -p 9870:9870 hadoop-namenode`? Explain each parameter's purpose.

Answer:

- `docker run` → launches a new container.
- `-d` → runs in detached (background) mode.
- `--name namenode` → assigns a readable container name.
- `--net hadoop-net` → connects to the specified Docker network (for inter-container communication).
- `-p 9870:9870` → maps host port 9870 to container port 9870 (NameNode web UI).
- `hadoop-namenode` → specifies the image to run.
  This command starts the Hadoop NameNode service inside a container accessible via browser at localhost:9870.

Q23. In what way does *MapReduce* achieve fault tolerance without user intervention?

Answer:
Each task writes intermediate results to local disk. If a worker fails, the ApplicationMaster reassigns its uncompleted tasks to another node using replicated input data.
Since MapReduce jobs are deterministic, rerunning a failed map or reduce produces identical results. This transparent re-execution model makes MR resilient without programmer-managed checkpoints.

Q24. Why is the Linux filesystem's single-root hierarchy (`/`) better suited for distributed systems compared to Windows-style drives?

Answer:
Linux's unified tree structure (`/bin`, `/usr`, `/home`, etc.) ensures consistent path references across all machines. Scripts can operate seamlessly regardless of device mounts.
In distributed systems, this simplifies container mounts, remote file access (via `/mnt` or `/data`), and Docker volume mapping. Windows' drive-based system (C:, D:) breaks portability, making automation less predictable.

Q25. Why is Hadoop's design philosophy based on "moving computation to data" rather than "data to computation"?

Answer:
Transferring terabytes across a network consumes huge bandwidth and time. Instead, Hadoop deploys code (kilobytes) near data (gigabytes).
Map tasks run on the same node holding HDFS blocks — leveraging local disk speed and reducing network bottlenecks. This *data locality principle* is what enables Hadoop to scale linearly even on low-cost commodity hardware.

✅ **End of Hard Exam Paper**