Assignment No. 02

Rubric	*Score (0 to 4)*
Delivery	
Understanding	
Readability	
Discipline	
Total	

Performed On: 11/03/25

Sign: _____

Q.1] Explain different load estimation and process transfer policies used by load balancing algorithms.

Ans. Load balancing ensures that computational tasks are evenly distributed across multiple nodes in distributed system to prevent bottlenecks and improve efficiency. Load balancing algorithms are different load estimation policies and process transfer policies to decides when and how to distribute tasks.

1. Load Estimation Policies :- Load estimation policies determine how the system measures and evaluates the workload of each node.

(i) CPU Queue Length - Based Estimation:
- Measuring number of processes waiting in the CPU queue.
- A higher queue length indicates a higher load.
- Simple but may not reflect the actual processing speed

(ii) CPU Utilization - Based Estimation:
- Measures the percentage of CPU usage over a given period.
- High CPU Utilization suggests an overloaded node
- Can be inaccurate if other resources.

(iii) Response Time - Based Estimation:
- Evaluates how long it takes for a process to execute
- High response time suggests system is under heavy load.
- Useful for interactive applications.

2. Process Transfer Policies:- Process transfer Policies determine when and which process should be transferred from an overloaded node to and understanding node

(i) Threshold - Based Policy:-
- Defines upper and lower threshold values for system load.
- If a node's load exceeds the upper threshold, it transfers processes to a lower-load node.
- If a load falls below the lower threshold, it can accept processes.

(ii) Load Difference - Based Policy:-
- Transfers processes when the difference between the most loaded and least loaded node exceeds a predefined threshold.
- Ensures and even distribution of tasks.

(iii) Probabilistic Transfer Policy:-
- Instead of fixed thresholds, it randomly selects processes for transfer based on probability.
- Reduces decision-making overhead but may be inefficient in some cases.

(iv) Sender-initiated Vs receiver initiated Policy.
- Sender-initiated:- An overloaded node actively looks for an underloaded node to send processes.
- Receiver-Initiated: An underloaded node requests tasks from overloaded nodes.

Q.2] Explain any 5 data centric consistency model with examples?

Ans. A- data-centric consistency model defines how updates to a shared data store appear to different processes in a distributed system. These models determine the order and visibility of read and write operations to ensure consistency while balancing system performance. Here are five key consistency models with examples:-

1. Strict Consistency :- Ensures that any read operation returns the most recent write value. Every process observes the same order of updates in real-time.

Example:
- Process P1 writes $X = 10$ at time $t1$.
- Process P2 reads X at time $t2$.
- P2 must see $X = 10$ immediately, ensuring real time consistency.

2. Sequential Consistency :- Ensures all processes observe operations in same order, but not necessarily in real-time. The sequence of operations must be consistent across all nodes.

Example:
- P1 : $x = 10$, then $Y = 20$
- P2 : Reads $Y = 20$, then reads $x = 10$
- P3 : Must see $x = 10$ before $Y = 20$

3. Causal Consistency :- Ensures that causally related operations appear in the correct order. Unrelated operations can appear in different orders from different processes.

Example:-
- P1: Writes $x=10$
- P2: Reads $x=10$, then writes $y=20$
- P3: Must see $x=10$ before $y=20$, but another write ($z=30$) updates to x and y can appear in any order.

4. Eventual Consistency:- Ensures that if no updates occur, all nodes eventually converge to the same value.

Example:- A user updates their profile picture on a social media platform.

5. Linearizability:- A stricter form of sequential consistency where operations appear instantaneously across all nodes.

Example:-
- P1: writes $x=10$ at time $t1$.
- P2: Reads x at time $t2 > t1$ and must see $x=10$ immediately
- P3: Cannot see an old value ($x=5$) after P2 has seen $x=10$.

Q.3] what are the desirable features of good DFS?

Ans. A Distributed file System (DFS) allow users to access and manage files across multiple networked machines as if they were stored locally. A good DFS should have the following desirable features:

1. Transparency:-
   - Access transparency: users should access remote files as if they were local
   - Location Transparency: file location

should not affect ~~the~~ users operations.

- Replication Transparency
- failure Transparency.

2. Scalability:- It should support dynamic addition of storage nodes.

3. fault Tolerance and Reliability:-
   - Data replication ensures availability even if a server fails.
   - Redundant storage minimizes data loss risks.

4. Performance Efficiency:-
   - fast file retrieval and minimal network latency.
   - Load balancing mechanism should prevent bottlenecks.

5. Security and Access Control:-
   - Authentication and authorization to control access to files.
   - Encryption for secure data transmission.

6. Ease of Use and Manageability:-
   - User-friendly interface for file operations
   - Easy integration with existing applications and operating systems.

7. Support for Heterogeneity:-
   - Support different file formats and access protocols (Eg. NFS, SMB, Hadoop HDFS).

Q.4] Explain various file caching schemes.

Ans. file caching is a technique used in distributed file system to improve performance by reducing access time and network traffic. Various caching schemes exist to optimize file access based on system requirements.

1. Client-Side Caching: The client machine store recently accessed files or file fragments locally to reduce the need for remote access.

advantage:-
- Reduces server load and network latency.
- Improves response time for frequently accessed files.

disadvantages:-
- Risk of stale data if files are modified at the server but not updated in cache

2. Server-Side Caching:- The server maintains a caching of frequently requested files to serve multiple clients efficiently.

advantages:-
- Reduces disk I/O operations.
- Improves performance for shared read operations.

disadvantages:
- Increased memory usage on the server
- May not benefit clients with highly personalized data access patterns.

3. Disk-Based Caching:- Instead of using memory cached files are stored on disk to handle large datasets.

advantage:-
- persistent caching even after system reboots.
- more storage capacity than RAM-based caching.

disadvantage:-
- slower than RAM-based caching due to disk read/write overhead.

4. Write-Through Caching:- Every write operation is immediately propagated to the server before being acknowledged.

advantages:-
- Guarantees data integrity.
- No risk of data loss due to cache failure.

disadvantages:-
- Higher latency due to frequent network communication
- slower performance compared to other caching schemes.

5. Write-Back Caching:- write operations are temporarily stored in the cache and written to the server only at specific intervals.

advantages:-
- Reduces network traffic and improves write performance.
- Useful for applications with frequent updates to the same file.

disadvantages:-
- May cause consistency issues if multiple clients modify same file.

Q.5] Explain Code migration.

Ans. Code migration refers to the process of moving program code, execution state, or both between different machines in a distributed system. This helps in improving performance, load balancing & resources utilization.

Types of Code Migration:-

1. Process Migration
   - Moves a running process from one machine to another
   - Useful for load balancing and fault tolerance.
   - Example: Migrating a running application from an overloaded server to less busy one.

2. Computation Migration:-
   - Moves only computation while data remains on the original machine.
   - Reduces network traffic by moving computation closer to the data.
   - Example: Running database query on the server instead of fetching all data to the client

3. Data Migration:-
   - Move data from one node to another for performance optimization.
   - Often used in cloud storage and distributed databases.
   - Example: Moving frequently accessed files to a faster storage system.