Answers written

Answers mille, but not done

1. Define Consistency Model. List its types.

2. List Desirable features of Good DFS.

3. Explain Domain Name System.

4. What are different types of load balancing algorithms?

5. Explain the Load balancing approach in distributed systems.

6. State the desirable features of the global scheduling algorithm.

7. Compare Load sharing and Load balancing strategies for scheduling processes in a

distributed system.

8. Discuss the issues in designing Load Balancing Algorithms.

9. Compare and contrast Task Assignment, Load Balancing and Load Sharing

approaches.

10.Explain the need of client centric consistency models as compared to data

centric consistency model. Explain any two client centric consistency models.

11.Write a short note on Andrew File system.

12.Explain Code Migration.

13.What do you mean by a consistency model? What are Different consistency

models?

14.Difference between data centric and client centric consistency models.

15.Explain Client centric and data centric consistency Models.

16.Discuss file caching for Distributed Algorithms.

17.Explain File Accessing Models.

18.What are the good features of a distributed file system? Explain File sharing

semantics of it.

19.Explain Name Resolution.

20.What are different types of failure? Which one is most dangerous?

21.What are different classes of failure in RPC?

# Module 4 Resource and Process Management

Intro:

* Distributed systems contain a set of resources that are interconnected by a network
* A resource could be logical (Shared file) or physical (CPU)
* A resource manager controls the assignment of resources to processes

Types of process scheduling techniques:

* Task assignment
* Load balancing
* Load sharing

### Desirable features of Global scheduling algo:

* No prior knowledge about the processes
  + The scheduling algo shouldn't require characteristics and resource needs of a process.
  + It reduces the burden on the user.
* Dynamic in nature
  + Process scheduling algorithms should always consider the current information system and should not obey any fixed static policy to assign the process to a particular node.
  + Load on nodes can change anytime. The scheduling algo should identify this change and migrate processes from one node to another if necessary.
  + Thus, the scheduling algo should be flexible
* Quick decision-making capability
  + Algorithms must make quick decisions about the assignment of tasks to nodes of the system.
* Balanced system performance and scheduling overhead
  + Process scheduling algorithms should not collect more global state information in order to use it for process assignment decisions. Otherwise there will be an increase in the cost of collecting and processing info. The information collected could be less important due to information aging. Hence, algo offering near optimal performance by using min global state info is preferred.
* Stability
  + Less of processing more of migrating.
  + Eg: Suppose node 1 and node 2 see that node 3 is idle. Both nodes transfer a part of their load to node 3 without co-ordinating because they act as independent nodes. If node 3 is overloaded, it transfers the processes to other nodes. This creates a cycle.
* Scalability
  + A scheduling algorithm should be capable of handling small as well as large networks.
* Fault tolerance
  + Should be capable of working after the crash of one or more nodes of the system.
* Fairness of service
  + More users initiating equivalent processes expect to receive the same quality of service.

### Task assignment

Assumptions:

* Processes have been split into tasks
* These are known
  + Computation requirements of tasks and speed of processors.
  + Resource requirements and available resources on node
  + Cost of processing tasks on nodes
  + Communication cost between every pair of tasks (IPC cost)

Basic idea: Finding an optimal assignment to achieve goals such as the following:

* Minimization of IPC costs

(IPC - Inter Process Communication)

IPC cost - Cost of communication between 2 tasks

* Quick turnaround time of process (turn around time - the amount of time taken to complete a process)
* High degree of parallelism
* Efficient utilization of resources

### Load balancing

It is a method to distribute workload among various processors in a network.

Goals:

* Min response time
* Efficient use of resources
* Avoid overloading
* Maximize throughput

Types:

1. Static

Takes into account average behavior of the system and ignores the current state of the nodes. Based on this, it will perform scheduling tasks.

1. Dynamic

Takes into account current info and constantly updates collected info if the state of a node is changed. Based on this info, it will perform scheduling tasks.

NOTE: Static is simpler but doesn't perform good when compared to dynamic.

Static

* Probabilistic
  + Uses info of the network topology, number of nodes, process capability, etc for decision making.
* Deterministic
  + Uses info about the properties of the nodes and characteristics of the processes for decision making.

Dynamic

* Centralized
  + A single node is responsible for scheduling. Due to which every other node passes info to this single node known as server node. The server node has system state info.
  + Drawbacks:
    - The server nodes receive lots of traffic which can be a bottleneck.
    - If server node crashes, the entire system stops
* Distributed
  + Many nodes are involved in the scheduling decision making. These nodes communicate with each other from time to time.
  + Cooperative
    - Distributed entities cooperate with each other.
  + Non cooperative
    - Entities act as autonomous ones and make scheduling decisions independently from other entities

Issues in designing load balancing algo:

* Need to determine workload of a node (Load estimation policy)
* Determine if process should be processed remotely or locally (Process transfer policy)
* Determine the number of times a process can be migrated (Migration limiting policy)
* Determine a way to share system state info among the nodes (State information exchange policy)
* Determine which node to send the current process to (Location policy)
* Determine the priority of execution of local and remote processes (Priority assignment policy)

Drawback of load balancing:

* Some system info needs to be stored which is not efficient since it generates a big overhead.
* It is not achievable because every node has a fluctuating number of processes assigned to it.

Load sharing:

* It is necessary and sufficient to prevent nodes from being idle while some other nodes have more than two processes.
* It is much simpler than load-balancing since it only attempts to ensure that no node is idle.
* Priority assignment policy and migration limiting policy are the same as that for the load-balancing algorithms.

### Compare Load sharing and Load balancing

| **Load Balancing** | **Load Sharing** |
| --- | --- |
| Load balancing is the idea of distributing the incoming traffic evenly across the nodes available such that all nodes have equal workload | Load sharing ensures that no node is idle if there is some other node with more than two processes |
| It is *less efficient* | It is *more efficient* |
| It is not necessary to prevent nodes from being idle | It is necessary to prevent nodes from being idle |
| It uses round robin fashion | It does not use round robin fashion |
| It is *complex* to set up and maintain | It is *easy* to set up and maintain |
| Eg. IGP | Eg. eBGP |

What is a process?

A process is an instance of a program running in a computer.

Process management includes making different policies and methods to share processors among all processes in the system (Best utilization of processor).

To achieve best utilization of processors:

* Process allocation
* Process migration
* Thread facilities

Process migration

* Non preemptive
  + Process is migrated (from source to new location) before execution in its current location.
* Preemptive
  + Process is migrated during its execution. The process has to be migrated along with the process environment.

Migration steps:

* Select a process
* Select destination node
* Move process

Advantages of process migration

* For reducing load on a node
* Higher throughput
* Efficient use of resources
* Minimum response time
* Replication increases system reliability

Types of process migration

* Homogenous
  + All nodes have the same architecture and OS.
* Heterogeneous
  + Migration is done from one node to the other node when the nodes have different machine architectures and OS.

Thread: A thread is the smallest unit of processing that can be performed in an OS. A single process can contain multiple threads.

Context switching: It is a process of storing the state of a process or thread, so that it can be restored and resume execution at a later point.

Process vs Thread

* Definition
* Context switching time
* Memory sharing (Process don't, threads do)
* Communication (Process to process, thread to thread)
* Resource consumption
* Dependency
* Time for creation
* Time for termination

<https://www.geeksforgeeks.org/difference-between-process-and-thread/>

Why process migration?

* Load balancing AND load sharing
* Fault tolerance
* Migrate processes closer to the source of some data

What is virtualization?

It is a process of creating a virtual version of a resource, program or device.

# Module 5 Consistency, Replication and Fault Tolerance

Why replication?

* Improve reliability, increase performance and for scalability.

How scalable?

* If there is an increase in the number of processes that access the data from the same server, then it is better to replicate the server to multiple servers and distribute the workload.

NOTE: You need to keep all copies consistent.

Consistency

Is an agreement between multiple nodes in a DS to achieve a certain value.

### Consistency model

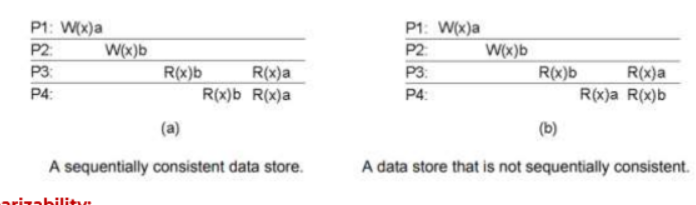
It is a contract between processes and the data store. If processes agree to follow certain rules the data store assures to work correctly.

#### Data centric

* + Agreeing on global order of updates on all the replicas.
  + No synchronization:
    - Strict consistency
      * All writes are visible instantaneously to all processes.



* + - Sequential consistency
      * The read operations should be in order of write operations (LIFO fashion).



* + - Causal consistency
      * Causally related writes
        + P1: W(x)a
        + P2: R(x)a W(x)b
        + Here, W(x)a and W(x)b are causally related because Process P2 reads 'a' first and uses it before writing 'b'.
        + So, the causal ordering should be:
        + P3: R(x)b R(x)a
      * Concurrent writes
        + P1: W(x)a
        + P2: W(x)b
        + Here, Process P2 does not use the value 'a' before writing 'b'.
        + Hence, they are not causally related so the ordering doesn't matter.
        + P3: R(x)a R(x)b
        + P4: R(x)b R(x)a
        + Both order of reading are correct
    - FIFO consistency (PRAM consistency)
      * Drop the causal ordering rule
      * Maintain ordering between multiple writes of the same process
      * Eg:
        + P1: W(x)a
        + P2: R(x)a W(x)b W(x)c
        + P3: R(x)a R(x)b R(x)c
        + P4: R(x)b R(x)a R(x)c
    - Linearizability
      * Not done
  + With synchronization:
    - Weak
      * Made consistent only after synchronization is done
    - Entry
      * Shared data is made consistent only when one process enters critical section
    - Release
      * Shared data is made consistent only when one process exits critical section

#### Client centric

* + - Environment
      * Most operations: "read"
      * “No" simultaneous updates
      * A relatively high degree of inconsistency tolerated (examples: DNS, www pages)
    - Wanted
      * Eventual consistency
      * Consistency seen by one single client
    - Eventual consistency
      * Only the process making the update should be able to see the update. But eventually, all the replicas need to be consistent. This can be done when no write operations are done by any process at some time.
    - Monotonic reads
      * “If process P reads the value of data item x, any successive read operation on x by that process will always return the same value or a more recent one.”
    - Monotonic writes
      * “A write operation by process P on data item x is completed before any successive write operation on x by the same process P can take place”.
    - Read your writes
      * The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.
    - Write follows reads
      * A write operation by a process on a data item x following a previous read operation on x by the same process is guaranteed to take place on the same or a more recent value of x that was read.

### Explain the need of client centric consistency models as compared to data centric consistency models. Explain any two client centric consistency models.

NEED OF CLIENT CENTRIC CONSISTENCY MODELS:

* System wide consistency is hard, so usually a client centric consistency model is more preferable.
* It is easier to deal with inconsistencies.
* Using a client centric consistency model, many inconsistencies can be hidden in a cheap way.
* It aims at providing a consistent view on a database.

### Types of client centric consistency models

* Eventual Consistency:

1. It is a weak consistency model.

2. It lacks simultaneous updates. (no w/w conflicts?)

3. It defines that if updates do not occur for a long period of time, all replicas will gradually become consistent

4. Eventual Consistency is a lot inexpensive to implement.

5. Requirements for Eventual Consistency are:

a. Few read/write conflicts.

b. No write/write conflicts.

c. Clients can accept temporary inconsistency.

6. Example: WWW.

* Monotonic reads

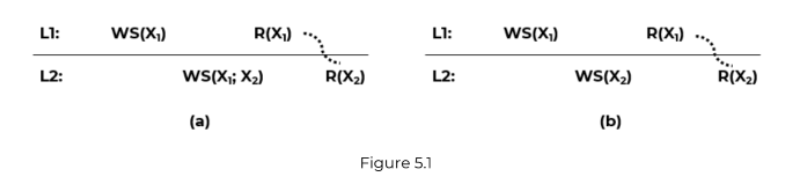
1. A data store is said to offer monotonic read consistency if the following condition holds:

“If process P reads the value of data item x, any successive read operation on x by that process will always return the same value or a more recent one.”

2. Consider a Distributed email database. It has distributed and replicated user-mailboxes.

3. Suppose the end user reads his mail in Mumbai. Assume that only reading mail does not affect the mailbox.

4. Later when the end user moves to Pune and opens his mail box again, monotonic read consistency guarantees that the messages that were in the mailbox in Mumbai will also be in the mailbox when it is opened in Pune.

  
[NOTE: Notation: WS(xi[t]) is the set of write operations (at Li) that lead to version xi of x (at time t). WS(xi[t1];xj [t2]) indicates that it is known that WS(xi[t1]) is part of WS(xj[t2])]

In the figure (a), process P first performs a read operation on x at L1 shown as R(x1), returning the value of x1. This value results from the write operations WS(x1) performed at L1. Later, P performs a read operation on x at L2, shown as R(x2). To guarantee monotonic-read consistency, all operations in W(x1) should have been propagated to L2 before the second read operation takes place. In other words, we need to know for sure that WS(x1) is part of WS(x2), which is expressed as WS(x1;x2).

In the figure (b), shows a situation in which monotonic-read consistency is not guaranteed. After process P has read x1 at L1. It later performs the operation R(x2) at L2. However, only the write operations WS(x2) have been performed at L2. No guarantees are given that this set also contains all operations contained in WS(x1)

Explanation:

Fig a.

Location L1: writes at x1, and then reads at x1

Location L2: wants to read at x2. But Mumbai Pune should read the same values. Whatever is read in Mumbai, should be replicated in Pune. Therefore WS(X1,X2) i.e. replication, and then read

* Monotonic writes  
  A data store is said to offer monotonic write consistency if the following condition holds:

“A write operation by process P on data item x is completed before any successive write

operation on x by the same process P can take place”.

1. Consider a software library example. In several cases, updating a library is accomplished by replacing one or more functions.

2. With monotonic write consistency, assurance is given that if an update is performed on a copy of the library, all previous or earlier updates will be accomplished primarily.



Fig a:

We want to write a google doc. We write some parts of it in Mumbai.

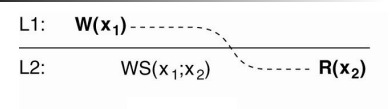
We later want to complete the remaining part of it in Pune. Hypothetically, think that the content written in Mumbai is replicated to a blank doc in Pune, to which the new content is later added by the user

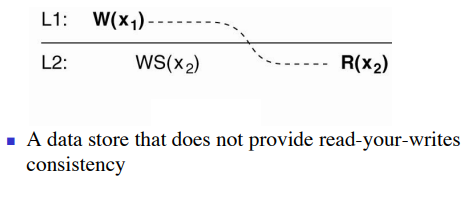
* Read your writes

A data store is said to provide read-your-writes consistency, if the following condition holds:

The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.

Whatever you are writing, same has to be read, no matter which location



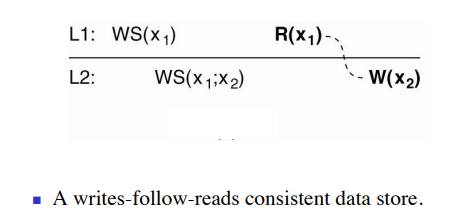
  
Example: Updating a Web page and guaranteeing that the Web browser shows the newest version instead of its cached copy.

* Writes follow reads

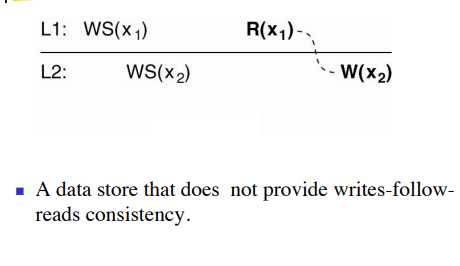
A write operation by a process on a data item x following a previous read operation

on x by the same process is guaranteed to take place on the same or a more recent

value of x that was read.

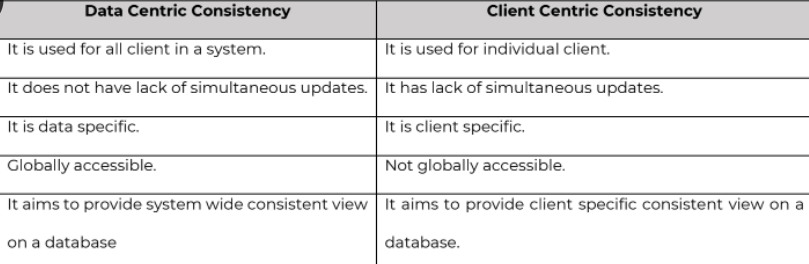


Update, and then write



Writes-follow-reads consistency can be used to guarantee that users can only post a reaction to an article only after they have seen the original article.

### Difference between Data Centric and Client Centric Consistency Models



Used: merchandise vs amazon

Lack of simultaneous updates: sequentially vs eventually

Accessible: merchandise vs amazon

Consistent: merchandise vs amazon

# Module 6 Distributed File Systems and Name Services

DFS

file storing on server, deleting editing then to manage that file, how do servers do all these? managing files on the server is done by the DFS. It is a client based application which gives permission to access and process data which is stored on the server .

### Desirable Features of good DFS

1. Transparency: hide information from user
2. Structured Transparency:   
   When a user tries to store a file on the server, multiple copies of the file are stored on multiple servers.   
   But the user should not know the number or locations of file servers or the storage devices instead it should look like a conventional file system (Centralize System)
3. Access Transparency:

The local and remote files should be accessible in the same way.

File system interface should not be able to distinguish between the local and remote files.

1. Naming Transparency:

Name of the file should not give any hint of the location of the file.

Without changing the filename, the file should be allowed to move from one node to another.

1. Replication Transparency:

If a file is located in multiple nodes, its existing multiple copies should be hidden from the client.

1. Scalability  
   A good DFS should be designed to easily cope up with the growth of the nodes, users of the system, and integration of resources
2. High availability (replication)  
   If any node fails or any storage device crashes, the DS should still work like normal.   
   Highly available DFS should have multiple independent file servers  
   NOTE: related to server crashing or data store crashing
3. High reliability  
   Probability of losing stored data should be minimized.   
   System should automatically keep a backup copy of files.  
   NOTE: related to data
4. Data integrity  
   Concurrent access requests from many users who are competing for access to the same file must be correctly synchronized using a concurrency control method
5. Security  
   In a DFS, there are many users that can store and access files.   
   An authenticated user should be able to access their specific files only.
6. User mobility  
   The user should have the flexibility to work on different nodes at different times. This can be achieved by automatically bringing the user's environment to the node where the user logs in.
7. Heterogeneity  
   A good DFS should provide the flexibility to the users to use different computer platforms for different applications
8. Performance  
   Performance is the average amount of time required to satisfy the client requests.  
   Performance of DFS should be comparable to the performance of centralized file system
9. Fault tolerance  
   Continue to provide correct service in the presence of communication or server faults

### File sharing semantics of DFS.

1. Unix semantics
   1. Absolute time ordering is performed on all operations. (FIFO?)
   2. UNIX semantic is easy to serialize read/write operations. (Sequential?)
2. Session semantics
   1. It consists of following access pattern:

i. Client opens a file.

ii. Performs read/write operations on file.

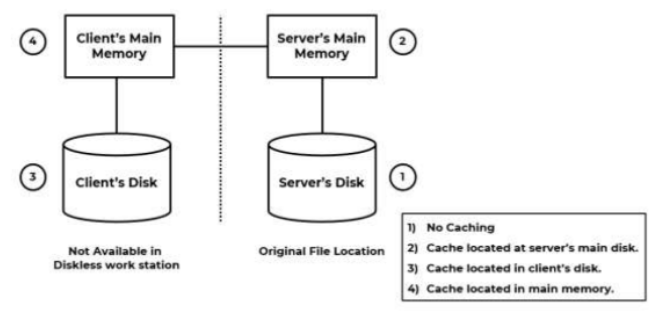
ii. Closes the file.

* 1. A session is a series of accessed files between open and close operations.
  2. In session semantics all changes made to the file are initially visible only to the client and invisible to other remote processes.

1. Immutable shared files semantics
   1. It is based on the use of an immutable file model.
   2. An immutable file cannot be modified once it has been created.
2. Transaction like semantics
   1. It is based on the transaction mechanism.
   2. A transaction is a set of operations between a pair of begin-transaction and end-transaction. (equivalent to session semantics?)
   3. It is a high-level mechanism for controlling concurrent access to shared mutable data.

### File caching for Distributed Algorithms

* File caching is implemented in a DFS to improve the performance of a DS
* File caching: retain recently accessed data in the main memory. Thus increasing the efficiency of accessing the same data multiple times.
* While implementing caching, one has to make several key decisions like granularity of cached data.
* 3 design issues
  + Cache location
    - Where cache data is stored
    - 3 possible locations: server’s main memory, client’s disk and client’s main memory



1. Server’s main memory: Reduces disk access cost
2. Client’s disk: Reduces network access cost, requires disk access cost
3. Client’s main memory: Reduces network and disk access cost
   * Modification propagation
     + A file’s data may be simultaneously cached on multiple client nodes
     + When one of the clients changes the file data, the corresponding data cached at other nodes should be changed or discarded
     + Questions
       - When to propagate modifications made to a cached data to corresponding file server?
       - How to verify the validity of cached data?
     + Answers
       - Write through scheme: Whenever cached data is modified, the modified data is passed to the master copy of the file
       - Delayed write scheme
         * When the user makes changes to the cached file, the changes are not propagated to the server immediately
         * Approaches

Write on ejection from  
Modified data in the cache is sent to the server when the cache replacement policy has decided to eject it from the client’s cache.

Periodic write

Write on close

* + Cache Validation Schemes
    - A file data resides in the cache of multiple nodes.
    - Check if the data cached at the client node is consistent with the master node.
    - If it is not then the cached data must be invalidated and the updated version must be fetched from the server.
    - Validation is done in two ways:
      * Client initiated approach
        + Client contacts the server to check if locally cached data is consistent with the master copy
        + Two types

Check before every access (defeats the caching purpose)

Periodic check

* + - * Server initiated approach
        + Server makes sure that the file (which is accessible to multiple clients) does not go in inconsistent state
        + Client informs the file server when it opens a file indicating whether the file is being opened for reading, writing or both.
        + The file server keeps a record of which client has which file opened and in what mode.
        + When a client closes a file it sends intimation to the server along with any modification made to the file.
        + On receiving intimation the server updates its record of which client has which file open in what mode.
        + Inconsistency occurs when more clients try to open a file in conflicting modes. Example: If a file is open for reading, other clients may be able to open it to read a file without any problem, but not for writing.
        + The server won't allow clients to open files in conflicting modes

### File Accessing Models

* File accessing model defines the service provided to clients request for accessing a file
* FAM depends
  + Unit of data access
  + Method used for accessing remote files

1. Unit of data access

Refers to fraction of file data that is transferred to and from the client as a result of single r/w operation

Four data transfer models

* File level transfer model
  + Whole file is transferred
  + Advantages
    - Scalable, Network is accessed only once, reduced server load and network traffic
  + Drawback: Requires sufficient storage space
  + Egs. CFS, AFS
* Block level transfer model
  + Units of data blocks
  + Contiguous portion of file of fixed length
  + Advantages: Does not require large storage space
  + Drawback: More network traffic and more network overhead
* Byte level transfer model
  + Unit of bytes
  + Maximum flexibility
  + Difficulty in cache management
* Record level transfer model
  + Suitable with structured files
  + Unit of records

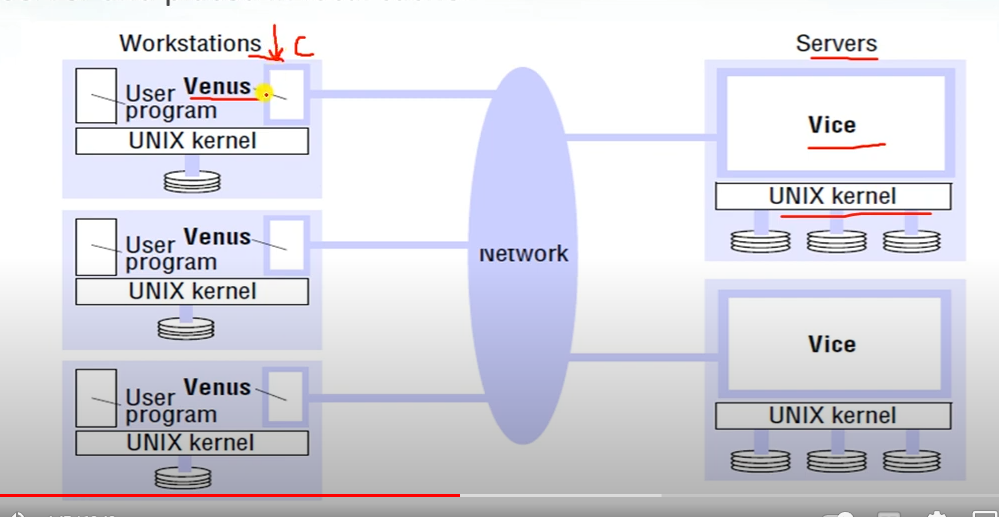
1. Accessing remote files

* Remote service model
  + Client - server model
  + Processing of client requests is performed at the server's node.
  + Client request is delivered to server and server machine performs it and returns replies to client.
  + Requests and replies transferred across the network as messages.
  + File server interface and communication protocol must be designed carefully so as to minimize the overhead of generating the messages.
  + Every remote file access results in traffic.
* Data caching model
  + Reduce the amount of network traffic by caching data
  + If data is not present in the cache, then it is copied from client node and cached
  + LRU is used to keep the cache size bounded
  + Cache consistency problem

### Naming Resolution

### Andrew File System (AFS)

* AFS is a DFS
* Developed by CMU
* Originally named “Vice”, AFS is named after Andrew Carnegie and Andrew Mellon
* Uses session semantics
* Supports info sharing on a large scale
* AFS is designed to (Features)
  + Handle terabytes of data
  + Handle thousands of users
  + Working in WAN environment
* AFS architecture
  + Two types of nodes
    - Vice node: Dedicated file server
    - Venus node: Client (local) machine
  + In AFS, when the file is opened, the entire file is copied from Vice node (server) to the Venus node (local machine)
  + When the file is changed and closed, it is copied back to the server

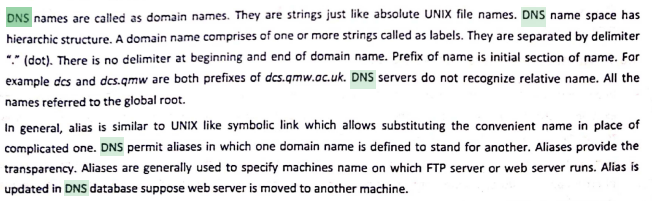


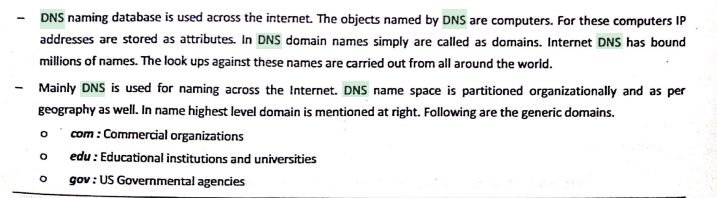
* Limitations
  + Non-availability of services when servers and network components fail.
  + Scaling problem.

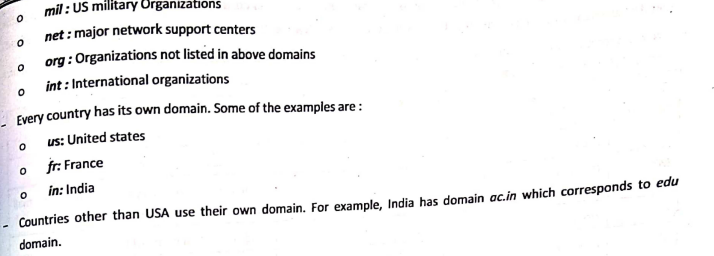
===========================================================

**Explain Domain Name system CHECK ANSWER**

* Hierarchic structure
* More than one strings called labels
* Delimiter “.” (not in beginning or end)
* Prefixes
* DNS servers do not recognize relative names. Global route

****

****

****

The Domain Name System (DNS) is the Internet's system for mapping alphabetic names to numeric Internet Protocol (IP) addresses like a phone book that maps a person's name to a phone number. For example, when a Web address (URL) is typed into a browser, a DNS query is made to learn an IP address of a Web server associated with that name.

Using the www.example.com URL, example.com is the domain name, and www is the hostname. DNS resolution maps www.example.com into an IP address (such as 192.0.2.1). When a user needs to load a webpage, a conversion must occur between what a user types into their web browser (www.example.com) into an IP address required to locate the www.example.com site.

Generic Domain :

- .com : commercial organizations

- Edu : educational institutions and universities

- gov : governmental agencies

- mil : military organizations

- int : international organizations

Country Domain :

- us : united states

- fr : France

- in : India