UNIT – III

PEER TO PEER SERVICES AND FILE SYSTEM

INTRODUCTION TO PEER-TO-PEER SYSTEMS

* In peer to peer n/ws, both resources and are widely distributed among nodes that are theoretically equals. Peer cooperate by sharing resources such as storage, CPU cycles, n/w bandwidth, and data.

* In a P2P s/m, participants (peers) cooperate to achieve a desired service. The service can be distributed computing, file-sharing, distributed storage, communication, or real-time media streaming. Ideally, there is no centralized entity to ctrl, organize, administer or maintain the entire s/m.

* characteristic of P2P:

1. Placement of data objects across many host.

2. Each participating m/c contributes resources

3. Better scalability for large no. of objects, due to distributed storage

4. Routes and object references can be replicated, tolerating failures of nodes

2. NAPSTER AND ITS LEGACY

* The first large scale peer to peer n/w was Napster, set up in 1999 to share digital music files over the internet

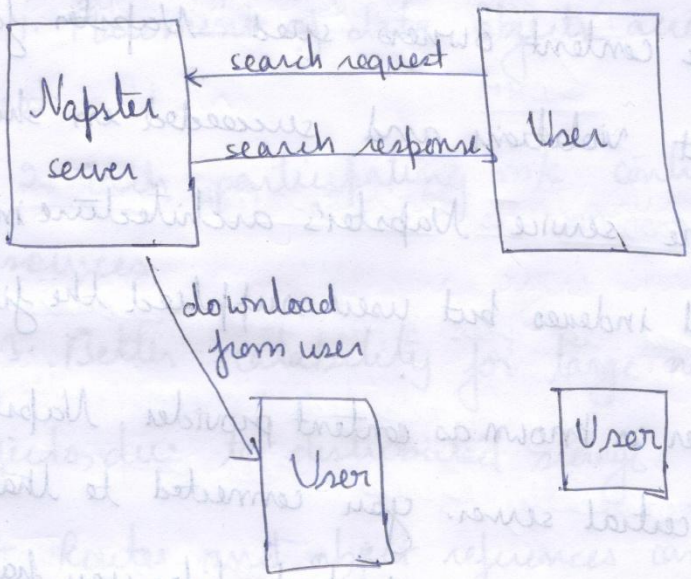
* while Napster maintained centralized and replicated indices, the music files were created and made available by individuals, usually with music copied

from CDs to computer files.

* Music content owners sued Napster for copyright violation and succeeded in shutting down the service. Napster's architecture included centralized indexes but users supplied the files

* Napster is known as content provider. Napster ran a central server. you connected to that server and announced what files you had to share. Each every search was conducted on the dataset assembled at the central server.

* Napster used centralized servers to keep a catalog of available files. Each peer needs to register itself to the central directory server before sharing its resources and querying the s/m, and no auto-discovery service is implemented in this s/m.



1. User sends out request Napster searches central database
2. The central server sends back a list of available files for download.
3. Requesting user downloads the file directly from another napster user computer.

* File downloading in a peer to peer point to point manner. The Napster

s/w tracks all users who are online at that particular time and provides access to tracks stored on user's hard drives.

* when the song ~~download~~ requested is found Napster establishes a connection b/w the two computers so that it could be downloaded.

Advantages:

1. Napster is simple
2. Files can be found fast and effective
3. Limited bandwidth usage

Disadvantage:

1. Central point of failure
2. Limited scale.

3. PEER TO PEER MIDDLEWARE

* Peer to peer middleware s/m's are designed specifically to meet the need for the automatic placement and

82

subsequent location of the distributed objects managed by peer-to-peer s/ms and applns.

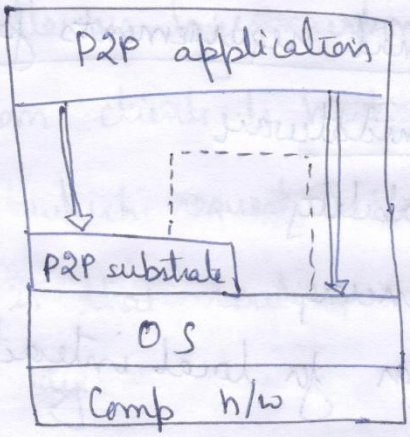
* The third generation introduced dedicated middleware layers which provided appln-independent mgmt of distributed resources

* The main objective are to:

1. Place resources on participating nodes that are widely spread over the internet
2. Route message to them on behalf of the client
3. Hide the location of resources from the client

* It also provides performance guarantee and place resources in a structured fashion to satisfy requirements of

availability, trust, load-balancing and locality.



* Functional requirements for peer-to-peer middleware

1. Allow clients to find and communicate with all individual resources that are made available to a service, even though the resources are widely distributed.

2. clients should be able to add new resources and remove them when they want to.

84

3. clients should be able to add hosts to the service and be able to remove them.

* Non-functional requirements for peer to peer middleware

1. Global scalability
2. Load balancing
3. Optimization for local interactions b/w neighboring peers
4. Accommodation to highly dynamic host availability
5. Anonymity, deniability and resistance to censorship.

4. ROUTING OVERLAYS

* A routing overlay is a distributed algm for a middleware layer responsible for routing requests from any client

to a host that holds the object to which the request is addressed.

* Routing overlay locates nodes and objects.
* is middleware layer responsible for routing requests from clients to host that holds the objects to which request is addressed. Main difference is that routing is implemented in applⁿ layer
* Task of overlay ~~of~~ routing
  1. Routing of requests to objects
  2. insertion of objects
  3. deletion of objects
  4. Node addition and removal
* A client is expected to be able to perform following task:
  1. Specify a GUID and an operation to the routing overlay. This sends the request to a live node with a replica of the

8b

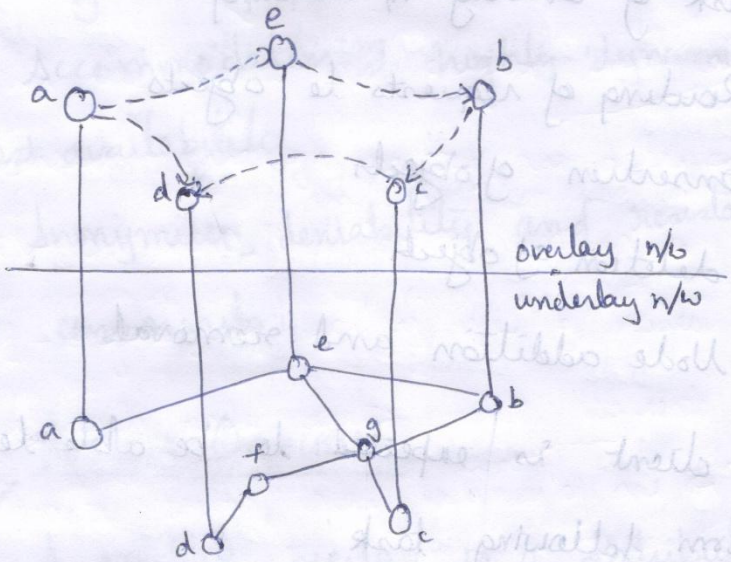
required resource.

2. provide a new resource after computing its GUID.

3. Remove a resource

4. Join or leave the n/w

* when a client requires to publish a resource, it has to compute the GUID and ask the routing overlay to publish it



87

* Basic programming interface for DHT

1. put (GUID, data)
2. remove (GUID)
3. get (GUID)

* Distributed object location and routing (DOLR)

1) Objects can be stored anywhere. DOLR layer maintains a mapping b/w GUIDs and the addresses of the nodes at which replicas of the objects are located.

2) DOLR layer routes the requests to the nearest available replica.

* Basic programming interface for DOLR.

1. publish (GUID)
2. unpublish (GUID)
3. SendToObj (msg, GUID, [n])

* Advantages of DHTs

1. DHTs scale and are ideal candidates for web scale storage.

88

2. They are more immune to node failures.

3. They use extensive data replication

5. OVERLAY CASE STUDIES : PASTRY, TAPESTRY

5.1 PASTRY

* PASTRY is an implementation of a Distributed Hash table algorithm for P2P routing overlay.

* All the nodes and objects that can be accessed through pastry are assigned 128-bit GUIDs. A Pastry s/m is a self-organizing overlay n/w of nodes, where each node routes client requests and interacts with local instances of one or more applns.

* All the nodes and objects are assigned 128-bit GUIDs.

1. For nodes: Computed by applying a secure hash function to public key of the node.

2. For objects: Computed by applying a secure hash function to the object name or some part of its stored state.

* Routing algm:

1. Stage 1: Simplified form of the algm which routes msgs correctly but inefficiently w/o a routing table.

2. Stage 2: Full routing algm which routes request to any node in O(log N) msgs.

* Each PASTRY node has a state consisting of:

1. Routing table used in the first phase of the routing

2. Neighborhood set M contains the NodeId and IP address of the |M| nodes which are

90

closet to the considered node.

3. Leaf set L contains the NodeId and IP address of the L/2 nodes whose NodeId are numerically closest smaller than the present NodeId, and the L/2 nodes whose NodeId are numerically closest larger than the present NodeId.

* Node departure or failure

→ A key design issue in Pastry is how to efficiently and dynamically maintain the node state, i.e, the routing table, leaf set and neighborhood sets, in the presence of node failures, node recoveries and new node arrivals.

→ A node is said to have failed when its immediately neighbors in the node ID space are no longer able to communicate with it

→ A failure in the routing table occurs when a node tries to contact another one for forwarding the msgs and gets no response

1. Failure in the leaf set
2. Failure in the routing table
3. Failure in the neighborhood set

5.2 TAPESTRY

* Tapestry is a decentralized distributed s/m. It is an overlay n/w that implements simple key based routing. each node serves as both an object store and a router that applns can contact to obtain objects. It also allows applns to implement multicasting in the overlay n/w.

* Tapestry routing tables

→ Tapestry supports a generic decentralized object location and Routing (DOLR) API using a self-repairing, soft-state based routing

92

layer

→ In order to allow nodes to locate objects stored at other nodes, each node maintains a routing table that stores references to a subset of the nodes in the n/w. A routing table has several levels; one level for each digit of the node's ID.

6. INTRODUCTION OF DISTRIBUTED FILE SYSTEMS

* File is a collection of data with a user view and a physical view.

* A Distributed File system (DFS) is a file s/m with distributed storage and users. DFS provides transparency of location, access and migration of files.

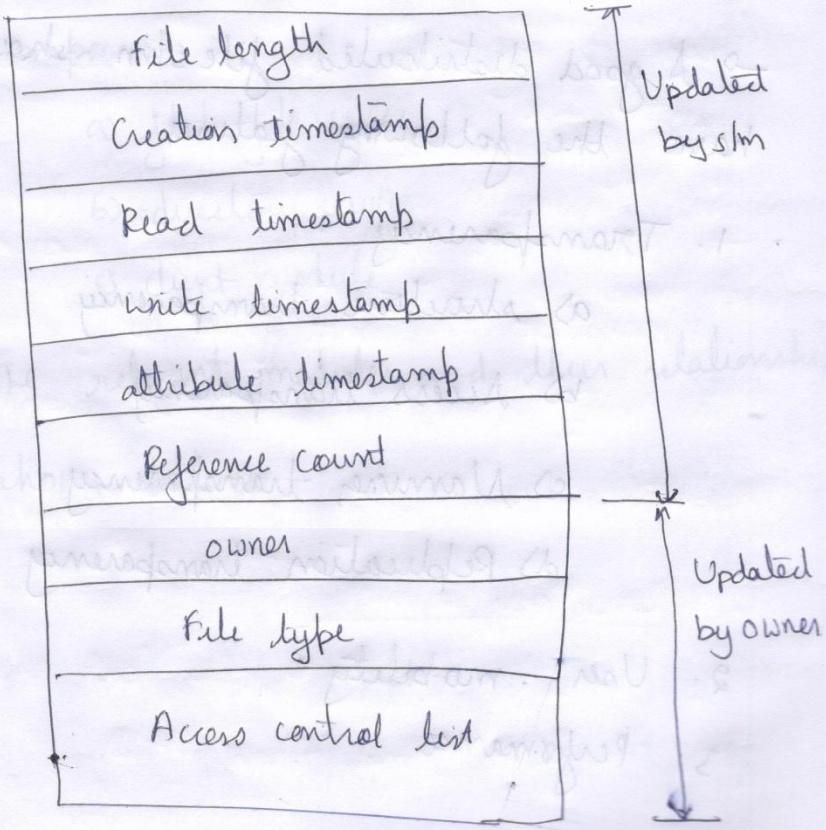
* A DFS enables pgms to store and access remote files exactly as they do local ones.

93

* Goals of distributed file s/m are as follows:
  1. n/w transparency
  2. High availability

* Characteristics of file s/m

→ File s/m is responsible for organization, storage, retrieval, naming, sharing and protection of files. Files are stored on disks or other non-volatile storage media.



94

→ File s/m module consists of following modules.

a) Directory module
b) File module
c) Access ctrl module
d) file access module
e) block module
f) device module

* Distributed File s/m requirement

→ A good distributed file s/m should have the following features:

1. Transparency
   a) structure transparency
   b) Access transparency
   c) Naming transparency
   d) Replication transparency
2. User mobility
3. Performance

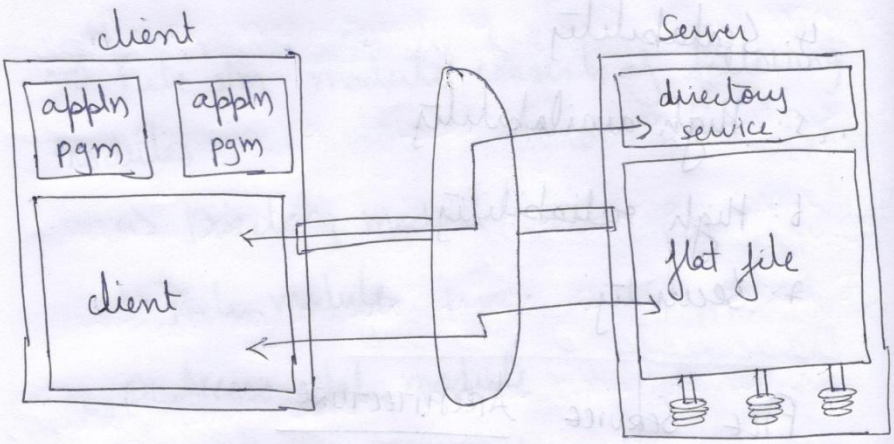
4. Scalability
5. High availability
6. High reliability
7. Security.

7. FILE SERVICE ARCHITECTURE

* An architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components:
  a) flat file service
  b) directory service
  c) client module

* The relevant modules and their relationships are shown below:

96



1. Flat file service:

→ concerned with the implementation of operations on the contents of file.

→ Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat file service operations

→ When the flat file service receives a request to create file, it generates a new UFID for it and returns the UFID to the requester.

2. Directory service:

→ Provides mapping b/w text names for the files and their UFIDs.

→ clients may obtain the UFID of a file by quoting its text name to directory service.

Client module:

→ It runs on each computer and provides integrated service as a single API to applⁿ pgms.

→ It holds information about the n/w locations of flat-file and directory server processes; and achieves better performance through implementation of a cache of recently used file blocks at the client

8) ANDREW FILE SYSTEM

* Andrew file s/m (AFS) is a distributed, networked file s/m that enables efficient file sharing b/w clients and servers.

AFS was originally developed for a

98

computer n/w running BSD UNIX and Mach.

* AFS is a global file s/m that appears as a branch of a traditional UNIX file s/m at each workstation. It is based on the client-server model and relies on RPCs for communication. There are three versions of AFS: AFS1, AFS2 and AFS3.

* In AFS1 and AFS2, the cache was updated only at the file level. Block level updates were not possible. It means that, users working locally on a file would not have to wait long for n/w access once the file is opened.

* AFS3 was also upgraded to support block level caching, which allows large files to be manipulated by clients with

98

computer n/w running BSD UNIX and Mach.

* AFS is a global file s/m that appears as a branch of a traditional UNIX file s/m at each workstation. It is based on the client-server model and relies on RPCs for communication. There are three versions of AFS: AFS1, AFS2 and AFS3.

* In AFS1 and AFS2, the cache was updated only at the file level. Block level updates were not possible. It means that, users working locally on a file would not have to wait long for n/w access once the file is opened.

* AFS3 was also upgraded to support block level caching, which allows large files to be manipulated by clients with

small caches.

* AFS server participates actively in client cache mgmt

* AFS has two unusual design characteristics:

1. whole-file serving
2. whole-file caching

* Implementations of AFS

→ The key s/w components in AFS are:

1. Vice:
2. Venus

* The basic file operations:

1. Open a file
2. Read and write
3. Close a file.

* System call interception in AFS

→ venus intercepts two s/m calls sent to the OS: open() and close().

9) File Model

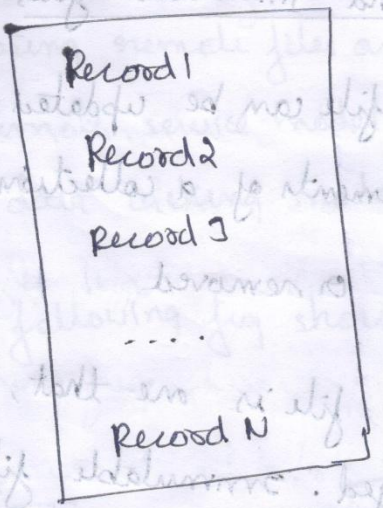
→ structure and modifiability are the criteria for file modeling. Different file s/ms uses different conceptual models of a file.

→ Unstructured and structured files

* In the unstructured model, file is an unstructured sequence of data. There is no substructure known to the file server. File content is also an un-interpreted sequence of bytes.

* In structured files, file appears to the file server as an ordered sequence of records. Records of different files of the same file s/m can be of different sizes.

~~* Structured files are of two types:~~
~~a) index record file~~



structured file format

* structured files are of two types
  a) index record file
  b) non-indexed record file.

* Index record file: Records have one or more key fields. key is used to access the records.

* Non-indexed record file: file record is accessed by specifying its position within the file.

→ Mutable and immutable files

* A mutable file can be updated or extended. This means elements of a collection can be changed, added or removed.

* An immutable file is one that, once created, cannot be changed. Immutable files are easy to cache and to replicate across servers since their contents are guaranteed to remain unchanged.

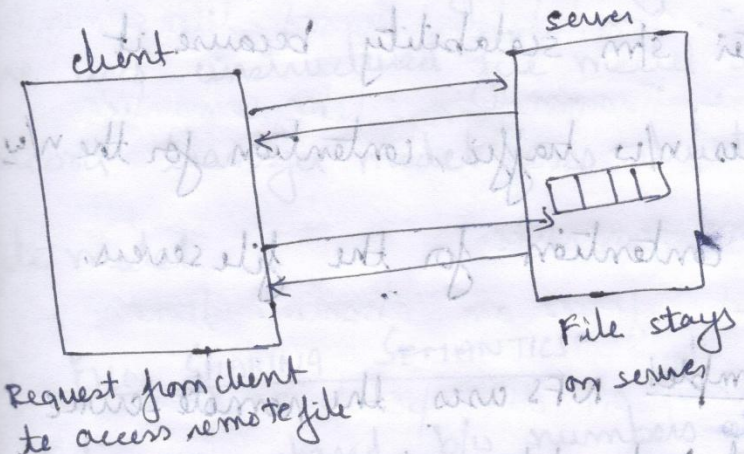
* Immutable file model eliminates all pblms associated with mutable versions of a file consistent. The immutable file model suffers from two potential pblms:

1. Increased use of disk space.
2. Increased a disk allocation m/g. activity.

10) File Accessing Model

* Accessing remote files are two types:
  1) Remote service model
  2) data caching model.

* The following fig shows remote access model



* Remote service model: client submits to server, all processing is done on server. File never moves from server.

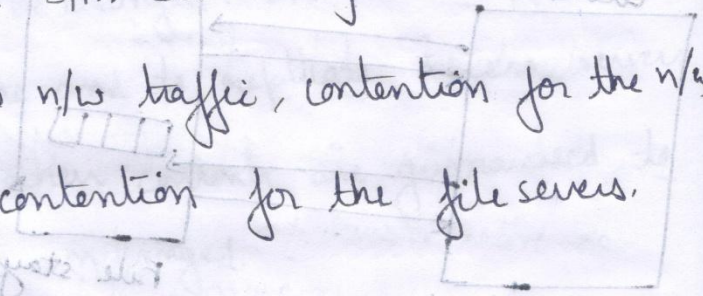
* Data caching model: It tries to reduce n/w traffic by using locality feature. for open file operation, transfer entire file to client and for close file operation,

104

it transfer entire file to server.

* Advantage of data-caching model over the remote service model:

→ The data-caching model offers the possibility of increased performance and greater s/m scalability because it reduces n/w traffic, contention for the n/w and contention for the file servers.



* Example: NFS uses the remote service model but adds caching for better performance.

* Unit of Data Transfer

→ Transfer levels are file, block, byte and record.

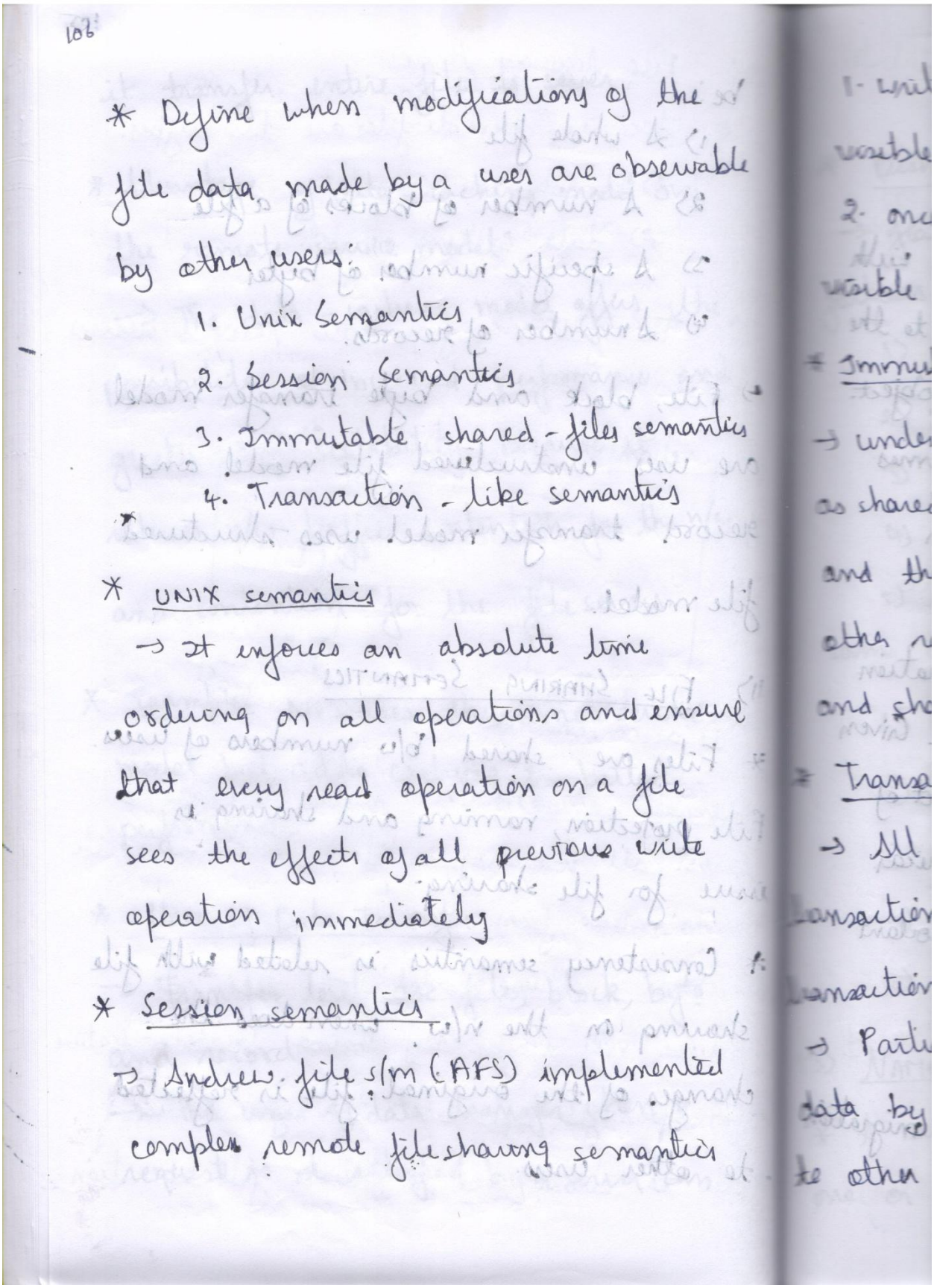
→ The unit of data transfer when a request is satisfied by a server can

be:
1) A whole file
2) A number of blocks of a file
3) A specific number of bytes
4) A number of records.

* File, block and byte transfer model are uses unstructured file model and record transfer model uses structured file model.

11) FILE SHARING SEMANTICS

* Files are shared b/w numbers of users. File protection, naming and sharing is issue for file sharing.
* Consistency semantics is related with file sharing on the n/w. when does the changes of the original file is reflected to other users.



106

* Define when modifications of the file data made by a user are observable by other users:

1. Unix Semantics
2. Session Semantics
3. Immutable shared-files semantics
4. Transaction-like semantics

* UNIX semantics

→ It enforces an absolute time ordering on all operations and ensure that every read operation on a file sees the effects of all previous write operation immediately

* Session semantics

→ Andrew file s/m (AFS) implemented complex remote file sharing semantics

1. writes to a file by a user is not visible to other users.
2. once the file is closed, the changes are visible only to new sessions.

* Immutable-shared-files semantics

→ under this s/m, when a file is declared as shared by its creator it becomes immutable and the name cannot be re-used for any other resource. Hence it becomes read-only, and shared access is simple.

* Transaction-like semantics

→ All changes occur atomically. Begin transaction, perform operation and end transaction

→ Partially modifications made to shared data by a transaction will not be visible to other concurrently executing transaction

until the transaction ends.

12) NAMING OVERVIEW

* A name in file s/m is associated with an object. Name resolution refers to the process of mapping a name to an object. Name space is a collection of names which may or may not share an identical resolution mech/-
* It provides a further level of abstraction when dealing with object replicas. Given an object name, it returns a set of the locations of the object's replicas. The naming s/m plays a very important role in achieving the goal of:
  1. location transparency
  2. facilitating transparent migration and replication of objects.

3. object sharing.

* **Desirable features of a good Naming s/m:**

→ Features of good naming s/m are as follows:

1) Location transparency
2) Location independency
3) Scalability
4) uniform naming convention
5) Multiple user-defined names for the same object.
6) Group naming
7) Meaningful names
8) Performance
9) Fault tolerance
10) Replication transparency

13) **NAME SPACE IMPLEMENTATION**

* A name service stores a collection of one or more naming contexts - sets of

bindings b/w textual names and attributes for objects.

* Major operation is to resolve a name to look up attributes from a given name.

other operations required:

creating new binding
deleting binding
listing bound names
adding and deleting contexts.

* Name mgmt is seperated from the services

a) Unification
b) Integration

* General Name service requirements

→ Handle arbitrary number of names and to serve arbitrary no of administrative organization.

a) A long lifetime
b) High availability
c) Fault isolation
d) Tolerance of mistrust

* Name spaces

→ A name space is a collection of all valid names recognizaton by a particular service. It allows simple but meaningful names to be used.

1) Flat name spaces.

→ The original set of m/c on the internet used flat namespaces

→ advantage

1. Names were convenient and short

→ Disadvantage

1. flat name spaces cannot generalize to large sets of m/c because of the single set of identifiers.

112

2 Single central name authority was overloaded.

3. Frequent name - address binding changes were costly and cumbersome.

iii) Hierarchical names

→ The partitioning of a namespace must be defined in such a way that it

i) supports efficient name mapping

ii) guarantees autonomous ctrl of name assignment.

→ Hierarchical namespaces provides a simple yet flexible naming structure.

→ The tree can have only 128 levels

→ Hierarchy of Name servers

* To distribute the information among many computers, DNS servers are used. creates many domains as there are first level nodes.

* In zone, a server is responsible and have some authority. The server makes database called zone file and keeps all the information for every node under that domain

→ Name Resolution

* DNS is designed as a client server application. A host that needs to map an address to a name or a name to an address calls a DNS client named a resolver.

→ Name servers:

* To avoid the pblms associated with having only a single source of information, the DNS name space is divided into nonoverlapping zones.

* When a resolver has a query about a domain name, it passes the query to one of the local name servers. If the domain being sought falls under the

114

jurisdiction of the Name server, it returns the authoritative resource records.

15) Name Caches

Characteristics:

a) High degree of locality of name lookup

b) slow update of name information db.

c) ~~one~~ on-use consistency of cached information is possible.

Types of Name caches

a) Directory cache

b) Prefix cache

c) Full name cache

Name cache implementation

Two mtds:

a) A cache per process

b) A single cache for all processes of a node.

Multicache consistency

* when a naming data update occurs, related name cache entries become stale and must be invalidated or updated properly

1. Immediate invalidate
2. On-use Update

ii) LDAP

* LDAP stands for lightweight Directory Access protocol. LDAP defines a std mtd for accessing and updating information in a directory
* LDAP is based on X.500. It is a fast growing technology for accessing common directory information
* LDAP uses:
  1. centrally manage users, groups and other data.
  2. Don't have to manage separate directories for each application.

Multicache consistency

* when a naming data update occurs, related name cache entries become stale and must be invalidated or updated properly

1. Immediate invalidate
2. On-use Update

ii) LDAP

* LDAP stands for lightweight Directory Access protocol. LDAP defines a std mtd for accessing and updating information in a directory
* LDAP is based on X.500. It is a fast growing technology for accessing common directory information
* LDAP uses:
  1. centrally manage users, groups and other data.
  2. Don't have to manage separate directories for each application.

116

3. Allow users to find data that they need.

4. Not locked into a particular server.

* LDAP is well suited for

1. Information that is referenced by many entities and applications

2. Information that needs to be accessed from more than one location

3. Information that is read more often than it is written

* LDAP is not well suited for

1. Information that changes often

2. Information that is unstructured.

* LDAP minimizes the overhead to establish a session allowing multiple operations from the same client session. LDAP defines

operations for accessing and modifying directory entries such as

1. searching for entries
2. Adding an entry
3. deleting an entry
4. Modifying an entry
5. Comparing an entry