UNIT - 2nd

DATA DELIVERY MODELS

With the explosion of Internet techniques and the popularity of mobile terminals such as laptops, personal digital assistants, people with battery powered MTs wish to access various kinds of web services over wireless networks at any time any place. However, existing wireless Internet services are limited by the constraints of mobile environments such as narrow bandwidth, asymmetric communication channels, unstable connectivity and limitations of battery technologies. The request-response type protocol can't scale up these problems. Data Dissemination is a type of protocol where server initiates and manages the transfer of data as well as updates. Data Dissemination also helps in maintaining data consistency and cache management. It entails distributing and pushing data generated by a set of computing systems and broadcasting data from audio, video and data services. The output data is sent to mobile devices. Then, mobile device can select, cache and tune required data items.

Communication Type

Symmetric Communication

⇒ Symmetric Communication is a type of communication in which transmission rates are same for both uplink & downlink.

⇒ eg⇒ GSM network.
Transmission rate of GSM network is 14.4 Kbps for both uplink & downlink

Asymmetric Communication

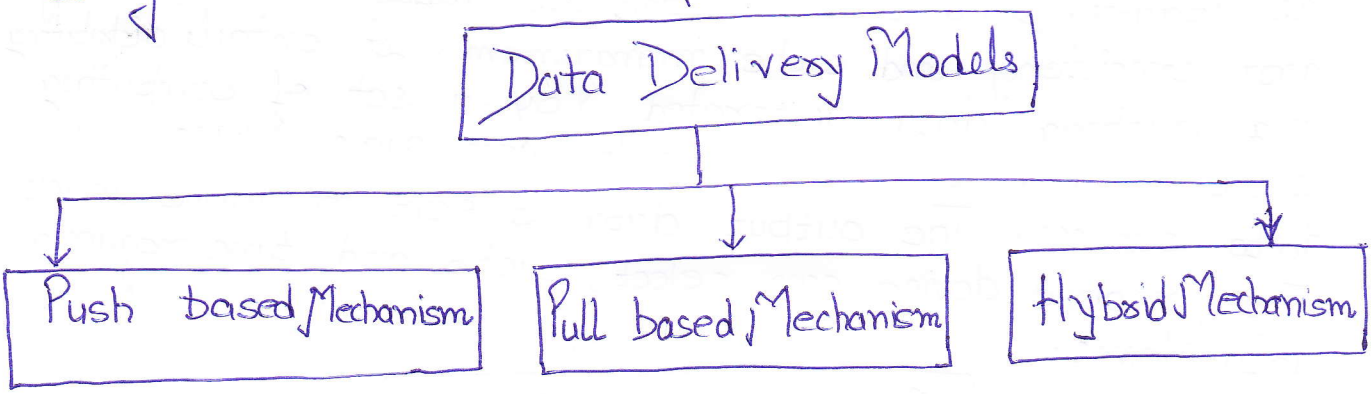
⇒ Asymmetric Communication is a type of communication in which downlink capacity is much greater than uplink capacity.

⇒ eg⇒ I-mode
downlink = 384 Kbps
uplink = 64 Kbps

Asymmetrical communication is best suited for mobile environments. In this architecture, there will be a stationary server continously broadcasting different data items. Mobile Clients continously listen to the channel and access the data of their interest. In mobile computing environment, large number of devices are attached or access the network. Bandwidth in the downstream from the server to the device is much greater than the upstream from device to server. This is because mobile devices have limited power resources.

Classification of Data Delivery Mechanisms.

Data delivery models can be classified into three categories on the basis of use.



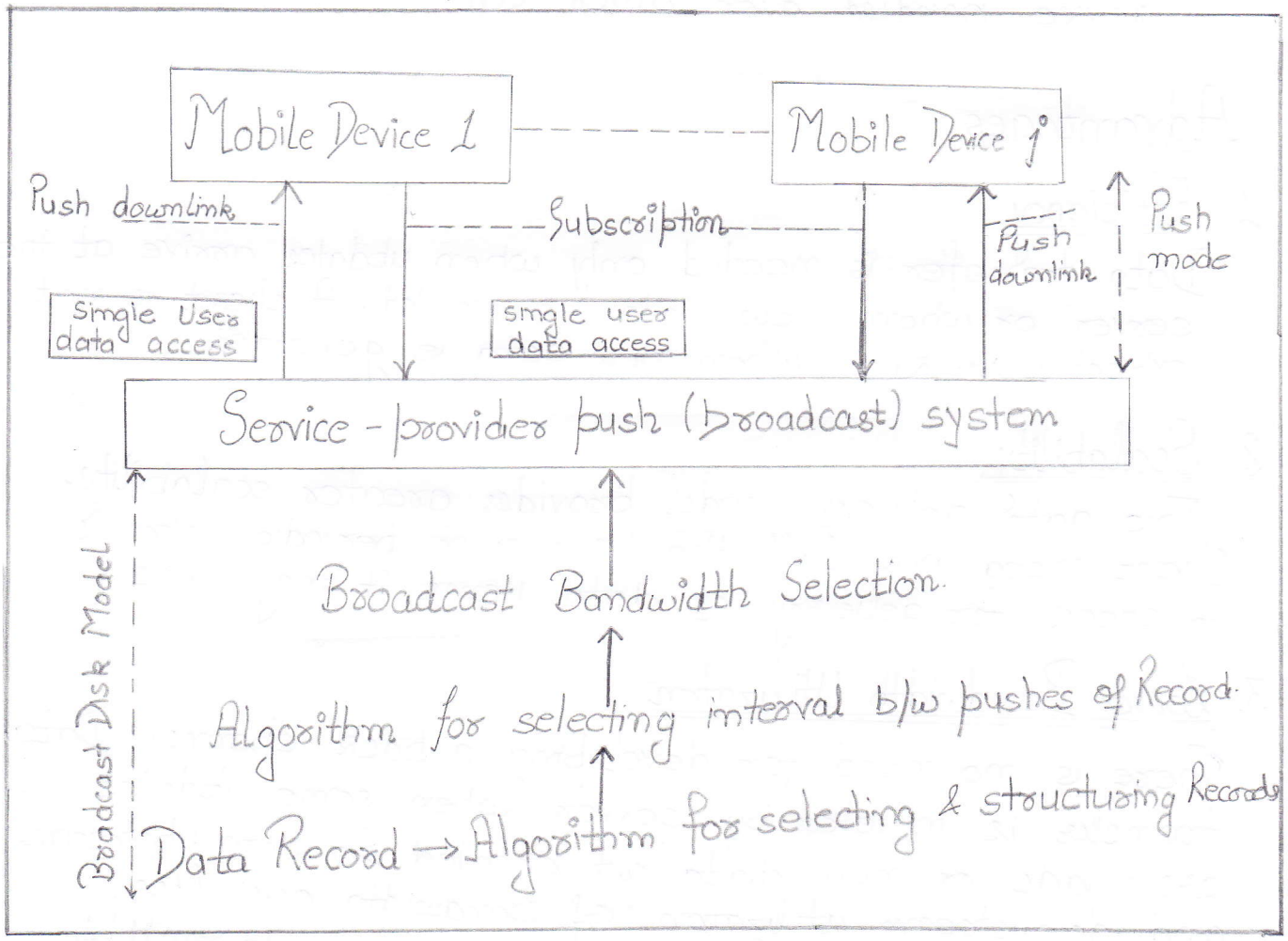
Push Based Model ⇒ Server disseminate information using periodic / aperiodic broadcast program.

Pull Based Model ⇒ Server disseminate information based on requests submitted by clients.

Hybrid Model ⇒ Push + Pull.

PUSH - BASED MODEL →

In push-based mechanism, the server broadcast data to all clients according to the broadcast program generated by data scheduling algorithm. The broadcast program determines the order and the frequencies that data items are broadcasted. Following figure shows a push based data delivery mechanism in which a server or computing system pushes data records from a set of computing systems. Data records are pushed to mobile devices by broadcasting without any demand. Also known as Publish-Subscribe mode.



Push-based mechanism works in following manner :—

1) Structure of data records to be published is selected. Algorithm provides an adaptable mechanism that permits data items to be pushed uniformly or non-uniformly after structuring them according to importance.
2) Data is pushed at selected time intervals using an adaptive algorithm.
3) Bandwidth are adapted for downlink (pushes) using an algorithm. Some fixed time can be used for pushing all records but usually higher bandwidth is allocated to records having higher number of subscribers.
4) Mechanism also adapted to stop pushes when a device handled over other cell.

Advantages

1. Efficiency
Data transfer is needed only when updates arrive at the server or when new data is created. A client is not expected to know when new data is generated.

2. Scalability
This data delivery model provides greater scalability. Client need not poll the server in periodic intervals to check for delievery of data items it may need.

3. Lower Bandwidth Utilization
There is no need for developing a back channel. Data transfer is initiated by server when some updates are made or new data get created. So both upstream and downstream utilization of bandwidth are less.

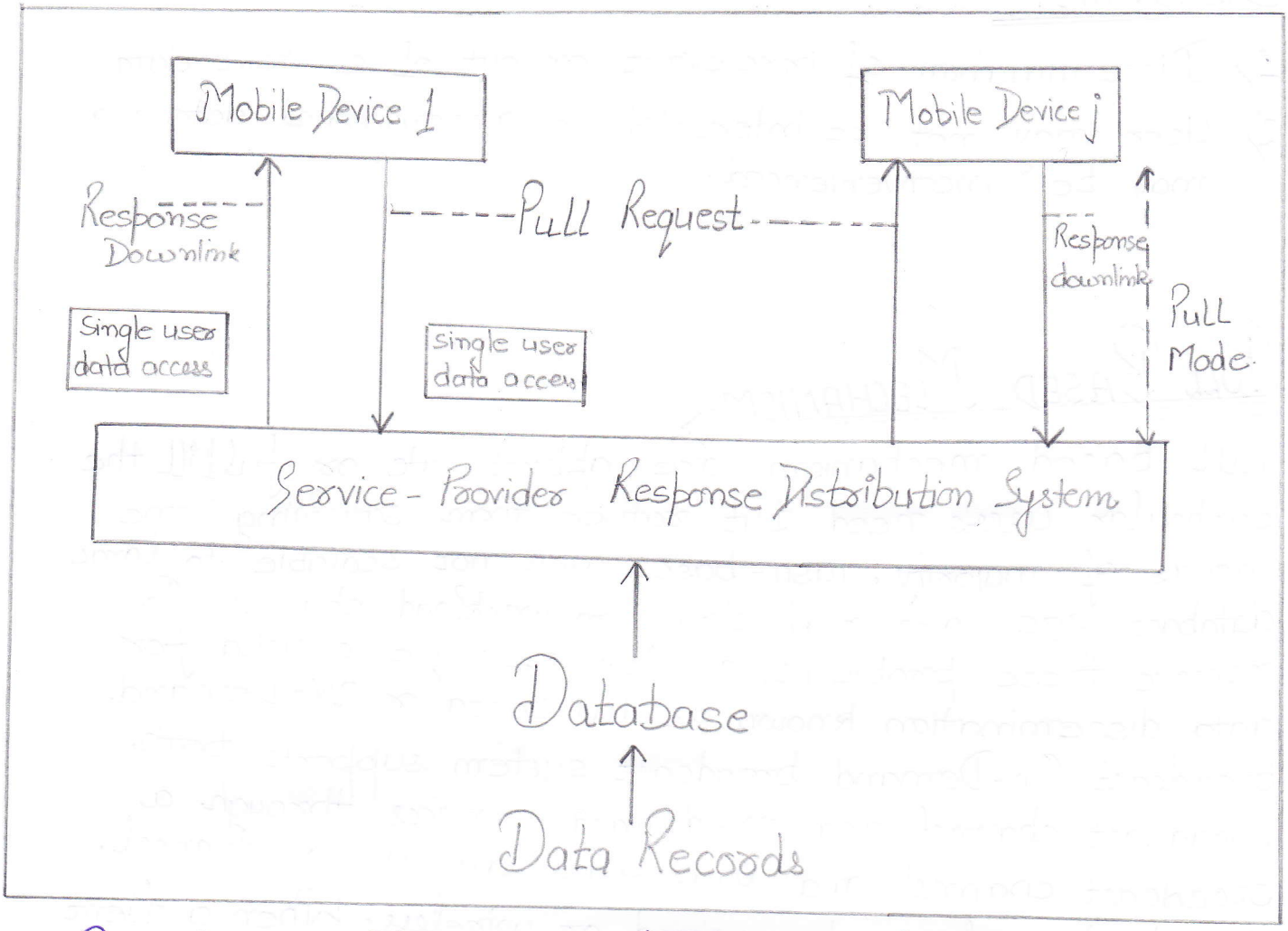
4. They enable broadcast of data services to multiple devices.

Disadvantage

1) Dissemination of irrelevant or out of context data.
2) User may not be interested in disseminated data and may be inconvenienced.

Pull Based Mechanism

Pull based mechanism are appropriate or fulfill the particular user's need but rather than satisfying the needs of majority. Push-based are not scalable to large database size and read slowly to workload changes. To remove these problems, a new technique is used for data dissemination known as Pull based or On-Demand broadcast. On-Demand broadcast system supports both broadcast ~~channel~~ and on-demand services through a broadcast channel and low bandwidth uplink channel. Uplink channel can be wired or wireless. When a client needs a data item, it sends to the server on demand request for the item through uplink. Client requests are queued up (if necessary) at the server on arrival. The server repeatedly chooses an item from outstanding requests, broadcasts it over the broadcast channel and removes the associated requests from the queue. Client monitors the broadcast channel and retreive them. Following figure shows a pull based mechanism in which a device pulls (demand) from a server or computing system. The data records are generated by a set of distributed computing systems. Examples of distributed server include music album server, ring tones server, video clips server, or bank accounting server.

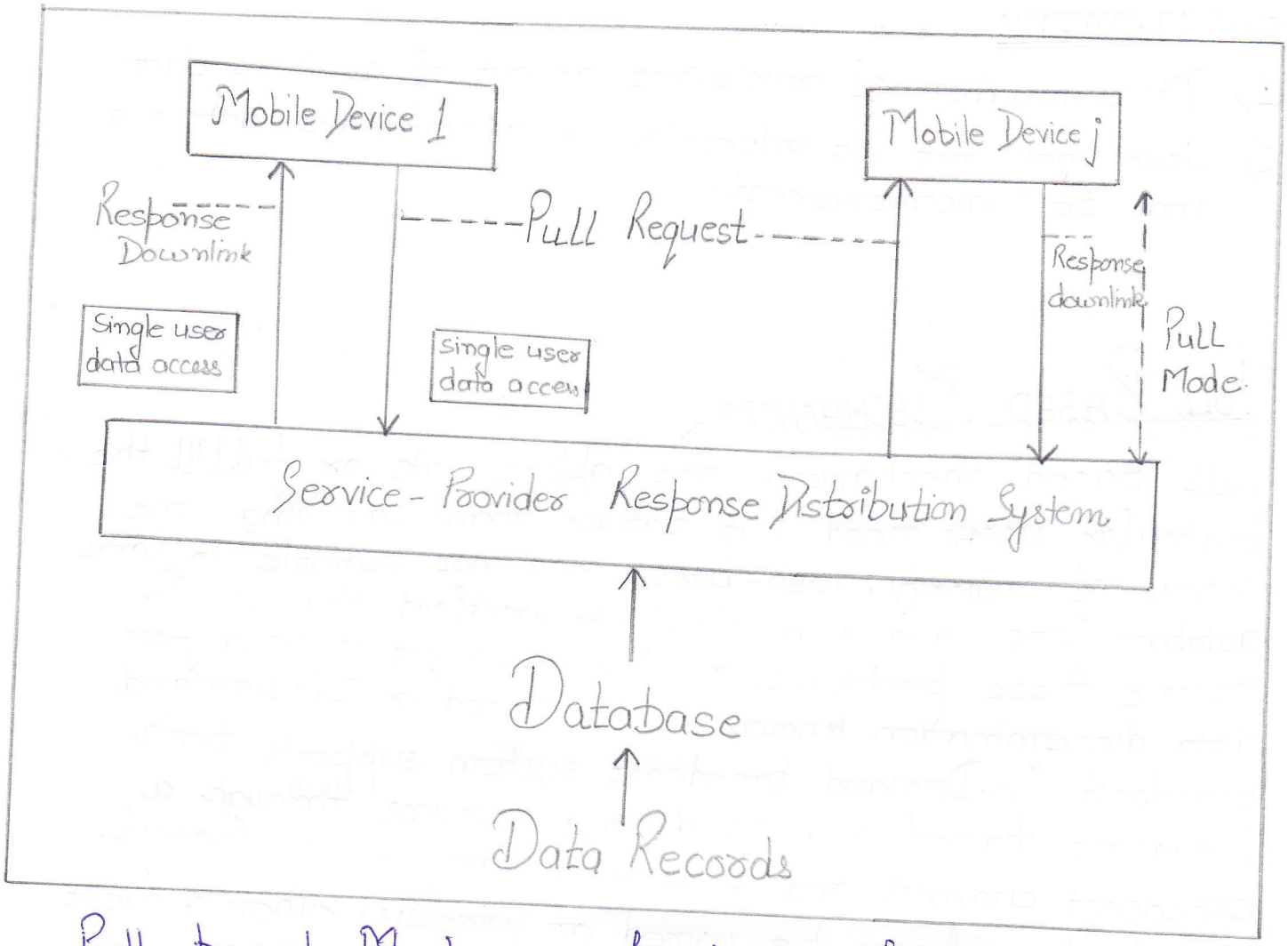


Pull-based Mechanism functions in following manner:-

1.) Bandwidth used for uplink channel depend upon number of pull requests. For example, uplink channel uses a bandwidth of 19.2 Kbps, and server accept accept 384 Kbps. Then only 20 pull requests can be handled. If number of pull requests is larger, the uplink channel may send requests at 9.8 Kbps or 4.8 Kbps.

2) A pull threshold is selected which limits the number of pull requests in a given period of time.

3) A mechanism is adopted to prevent device from pulling a cell, which has handled over device to another cell.



Pull-based Mechanism functions in following manner:-

1.) Bandwidth used for uplink channel depend upon number of pull requests. For example, uplink channel uses a bandwidth of 19.2 Kbps, and server accept accept 384 Kbps. Then only 20 pull requests can be handled. If number of pull requests is larger, the uplink channel may send requests at 9.8 Kbps or 4.8 Kbps.

2) A pull threshold is selected which limits the number of pull requests in a given period of time.

3) A mechanism is adopted to prevent device from pulling a cell, which has handled over device to another cell.

Advantages

1) No irrelevant data arrive at the device and only relevant data is disseminated only when user asks for it.
2) This model is best option when server has very little concern and only respond to many requests within expected time intervals.

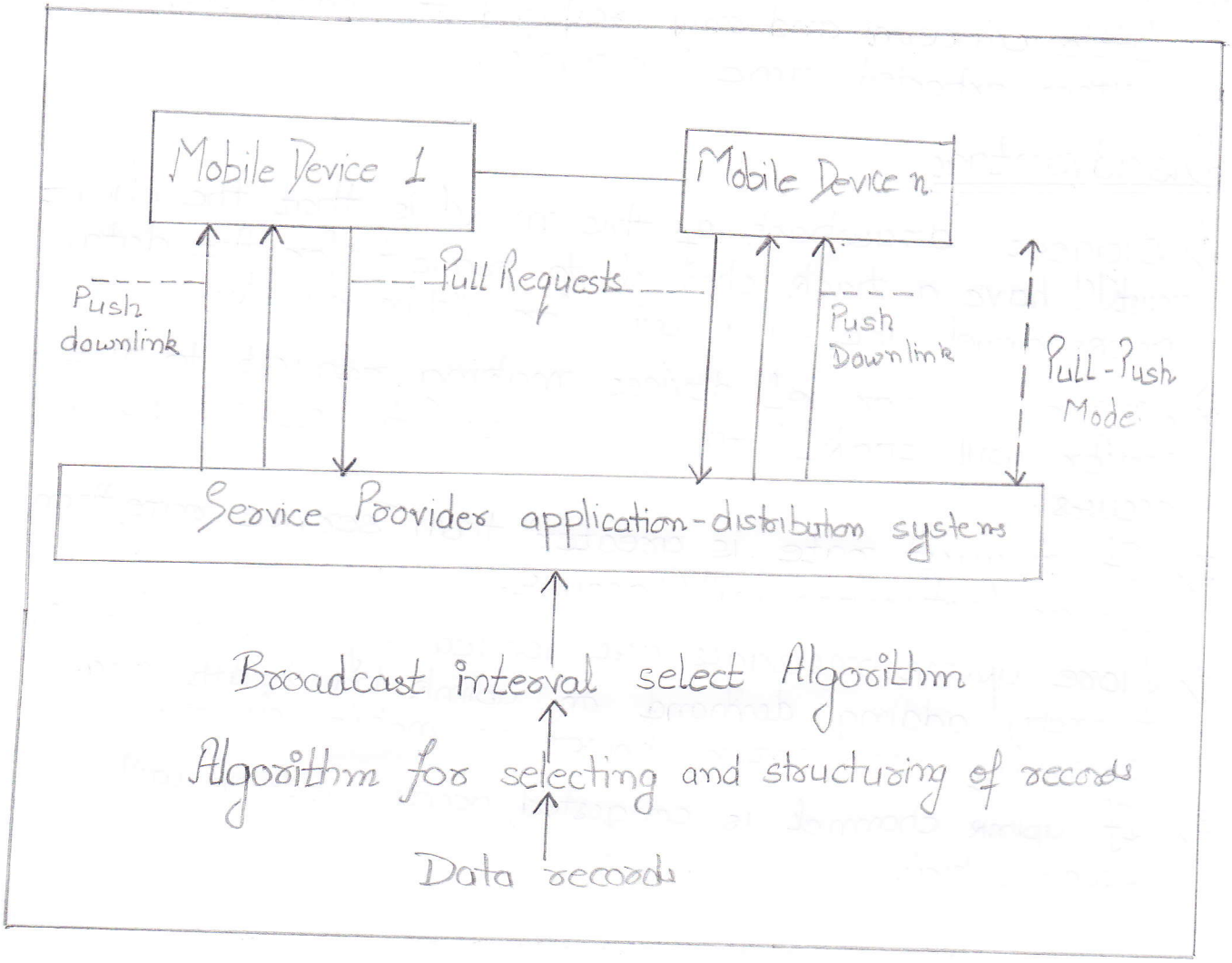
Disadvantage

1) Biggest drawback of this model is that the client must have a back channel to request for the data. Backchannel use bandwidth for data transfer.
2) Larger number of devices making request to the server will choke the network, hence congestion occurs.
3) If request rate is greater than service rate, then server saturation will occur.
4) More uplink messages are issued by mobile clients, thereby adding demand on uplink bandwidth and consuming more battery power on mobile clients.
5) If uplink channel is congested, access latency will become high.

Hybrid Mechanism

This method combine push or pull based mechanism so that they can complement each other. In this method, items are classified as frequently requested (frequest) or infrequently request (irequest). It is assumed that the client know which is frequest & which are irequests. Frequests using a broadcast cycle and irequest using on demand. For example, Advertising and Selling Music albums. Advertisements are pushed and mobile devices pull for buffering the albums.

Following figure shows a hybrid mechanism, in which a device pulls (demands) from a server and the server interleaves the responses along with the pushes of data records by a set of distributed computing systems.



Hybrid mechanism functions in the following way :-

1. There are two channels, one for pushes by front channel and other for pulls by back channel.
2. Bandwidth is shared and adapted between the two channels. It is adapted in downlink and uplink channels depending upon the number of active devices receiving data from server and the number of devices requesting data pulls from server.

DATA DISSEMINATION BY BROADCAST

Wireless environment provides that a device may not always remain connected and networked to the data server. Number of breaks occur in the connection. Therefore, the data records pushed from the server may be missed and not be cached. One way to overcome this problem is that the server can repeat pushing the records at successive intervals of time and use some specific algorithm or model for repeating each pushed record. In such case, when a device is connected at a certain instant with no breaks in its connection, it may tune to the selected record and cache it for its application.

A disk model in which it is presumed that all the n records to be broadcast are stored on a circular disk from 0° to 360° is called broadcast disk model. As the disk revolves and angle changes from 0° to 360°, the entire N bits in n records get pushed through a reading head over the disk. The head continously reads each bit of a record and broadcasts it instantenously on the wireless network. If a device misses a record in first revolution, it can cache the same in next or any of successive revolution.

$$\text{Broadcast Bandwidth} = \frac{\text{Number of bits } (N)}{\text{time taken for one revolution, } T_s}$$

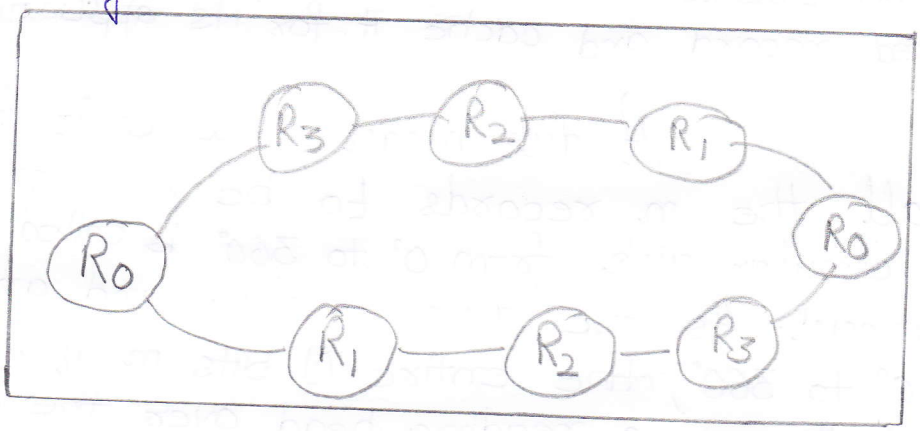
There are number of adaptions and algorithms for broadcast, each have its own advantages and disadvantages.

Algorithm for Broadcast

- Flat - Disk Model
- Multi Disk Model
- Skewed Disk Model

Flat Disk Model

The broadcast disk model is called flat disk model when all bits of all records are stored on and broadcast from one level only. In this model, each block of records is pushed after equal intervals of time. Following figure shows a rotating disk cycle with record blocks R_0, R_1, R_2, R_3, R_0, R_1, R_2, R_3 transmitted in a single broadcast cycle. All record blocks have an equal priority level. Using the flat-disk model, the server broadcasts the data as per cyclic requests without taking into account the number of devices that subscribe to particular record.

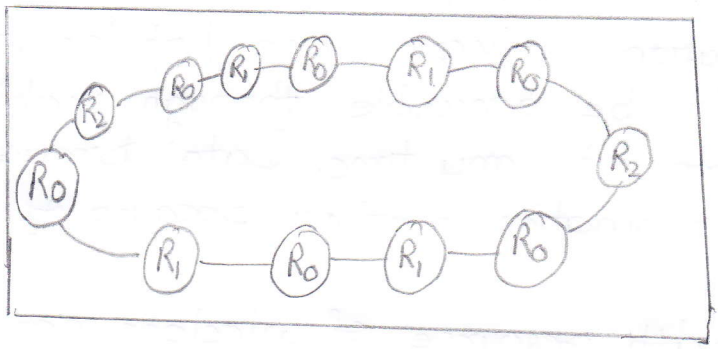


Multi-Disk Model

Multi-disk model is a type of broadcast model in which each block of records at a level is pushed in equal time T_s, but a block of higher priority is repeated more often at more levels. The multi-disk model have multiple levels of records on the broadcast disk. The number of levels assigned to a block of records depends on its priority for being pushed or the number of users subscribing to it. The transmission rate at each level is same but repetition rate of a block of records is proportional to the record's priority level. It is assumed that there are multiple disks, each with same pushing rate but different records in them. The high priority records are pushed more by multiple disks than lower one.

example Consider three records R_0, R_1, R_2 which are to transmitted during one broadcast cycle in order $R_0, R_1, R_0, R_1, R_0, R_2, R_0, R_1, R_0, R_1, R_0, R_2$.

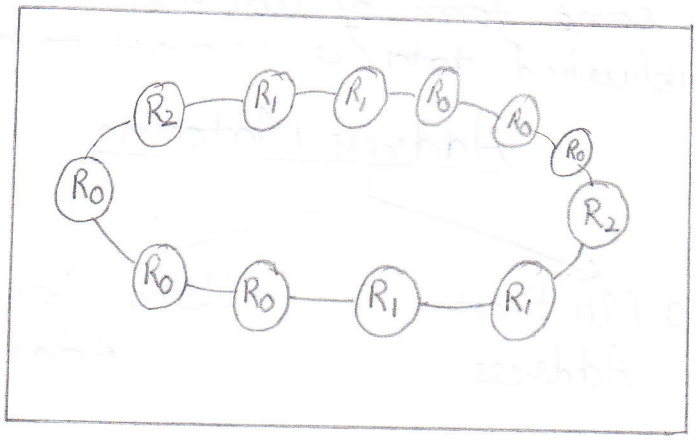
This order is repeated cyclically & time between two successive pushes of particular data remain same.



Skewed-Disk Model

In this model, the block of records are repeated as per their priorities for pushing or as per number of subscribers of a given record. High priority record blocks are pushed more often than the low priority ones. Unlike multi-disk model, skewed-disk model have consecutive repeated transmissions of a record block, followed by a consecutive repeated transmissions of another record block, and, so on. The number of transmissions of a record block depends upon its priority.

example R_0, R_1 and R_2 are three record blocks to be transmitted in single broadcast cycle and R_0 has highest priority, followed by R_1 and then R_2. These record blocks are transmitted in the order $R_0, R_0, R_0, R_1, R_1, R_1, R_2, R_0, R_0, R_0, R_1, R_1$ and R_2.



Organising & Publishing of DATA in AIR

The rapid development of wireless communication technology and battery powered portable devices making mobile information services more popular. A variety of information will be accessible through mobile devices from anywhere at any time. Data broadcast is an attractive dissemination method because of two main reasons :—

⇒ Bandwidth resource of wireless network is scare.

⇒ Most wireless systems are asymmetric in that the downlink communication capacity from base stations to the clients is much higher than uplink capacity.

The performance of broadcast dissemination based on latency and Access time. Our main goal is to minimize the access time. A periodic broadcast or multicast of data on wireless channel by a MSS to a specific group of mobile client is known as publishing in air. Publishing is suitable for asymmetric communication environment. There is virtually no data transfer from clients to the server in a dissemination based information system. The clients filter out published data which arrive on downlink channel and capture only a small set of relevant data for use by the application running on them. The process of filtering data requires some form of address matching to determine published data is relevant to a client.

Address Matching

- Using Multicast Address
- Using Temporal Address

Selective tuning is a process by which client device selects only the required records, tunes to them and caches them. Tuning means getting ready for caching at those instant and intervals when a selected record of interest broadcasts. Basic broadcast scheme is sequential. We want to minimize client's access time and tuning time. Consider the following :—

1. There are n records R_0 to R_{n-1} which are broadcast as in multi-disk model.
2. Only the records R_i' and R_j' which are of interest and required by applications at a device.
3. The broadcast disk broadcasts R_i' and R_j' thrice & once respectively.
4. The record R_i' is partitioned into k buckets such that b_{i_0} to $b_{i_{k-1}}$.
5. The record R_j' is partitioned into k' buckets such that b_{j_0} to $b_{j_{k'-1}}$.

⇒ Each bucket has equal length l_b which means equal number of bits and devices take identical time to cache each bucket data.

$$\boxed{\text{Time taken to cache the data} = L_b \times t_s}$$

t_s → time taken or interval between successive bits.

L_b → bucket length.

Access time ⇒ Represented by t_{access}

It is the time interval between pull request from device and reception of response from broadcasting or data pushing or responding server. Two important factors that affect t_{access} are :—

⇒ number & size of records to be broadcast

⇒ directory or cache miss factor

Selective Tuning Methods

- Directory Method
- Hash Based Method
- Index Based Method
- Distributed Index Method.
- Flexible Indexing Method
- Alternative Method.

1.) Directory Method

This method involves broadcasting a directory at the beginning of each broadcast cycle. If interval between the start of the broadcast cycles is T, then directory broadcasts at each successive intervals of T. A directory specifies when a specific record or data item appears in data being broadcasted. Directory contain start sign, pointers for locating records, end sign.

Directory = Start Sign + pointers for locating record + end sign

A device has to wait for the directory, then it has to wait for the required bucket or record before it can get tuned to it and start caching it.

Tuning Time Represented by t_{tune}. It is the time taken by the device for selection of records.

t_{tune} = time spent in listening to directory signs + pointers for the record to select bucket or record by device + Waiting for buckets

The device selectively tunes to the broadcast data to download the records of interest. When a directory is broadcast along with data records, it minimizes the t_{tune} and t_{access}. The device saves energy by remaining active just for the periods of caching the directory. For rest of time, it remains idle or performs application tasks.

⇒ Without use of directory for tuning, $t_{tune} = t_{access}$ and the device is not idle during any time interval.

⇒ Directory or cache miss occurs if t_{tune} is longer than T.

2. Hash-Based Method

Hash is a result of operations on pair of key & record. Advantage of broadcasting a hash is that it contains a fewer bits compared to key and record separately. The operations are done by hashing functions. From the server end the hash is broadcasted and from the device end a key is extracted by computations from the data in the record by operating the data with a function called hash function. Hash based method provides that the hash for the hashing parameter is broadcasted. Each device receives it and tunes to the record as per the extracted key. In this method, the records that are of interest to a device or those required by it are cached from the broadcast cycle by extracting and identifying the hash key which provides the location of the record. This helps in tuning of the device.

3) INDEX-BASED METHOD

When we search for a particular topic in a book, we search the index and find a number which tells the page number where that topic can be found. Similarly in a broadcast cycle, a number called index can be first sent. It specifies the location of the bucket or record. Thus, indexing is another method for selective tuning. Indexes map the location of the buckets. At each location, an offset value may be specified.

Index ⇒ Maps to the absolute location from the beginning of a broadcast cycle

Offset → Number which maps to the relative location after end of present bucket of interest

Indexing is a technique in which each data bucket, record, or record block of interest is assigned an index at previous data bucket, record or record block of interest to enable the device to tune and cache the bucket after the wait. The server transmits this index at the beginning of a broadcast cycle as well as with each bucket corresponding to data of interest to the device.

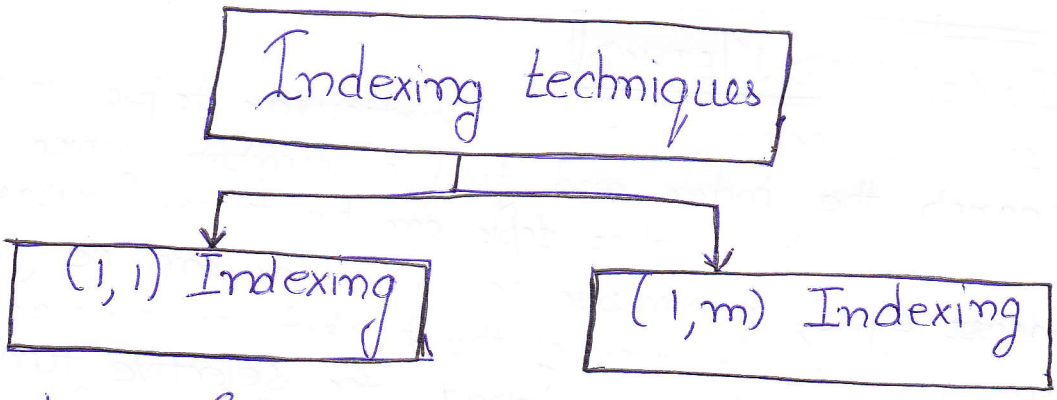
Advantage of having an Index

⇒ Device just reads it and selectively tunes to the data buckets or records of interest instead of reading all the data records.

⇒ During the time intervals in which data is not of interest, the device remains in idle or power mode.

Disadvantage

⇒ It extends the broadcast cycle & increases access.



(1, 1) Indexing Scheme

In this indexing scheme, the index I transmitted only once with every broadcast cycle. The access time for this scheme is given by:-

$$\boxed{\text{Access Time} = \text{Data} + \text{Index}}$$

Main problem with (1,1) indexing scheme is that it increases access latency because if an index is lost during transmission due to loss, then the device must wait for the next transmission of the same index-record pair. The data tuning time now increases by an interval equal to the time required for one broadcast cycle.

(1,m) Indexing Scheme

(1,m) Indexing scheme means an index I is transmitted m times during each transmission of record. An algorithm is used to adapt a value of m such that it minimizes access latency in a given wireless environment.

The index format is adapted to (1,m) with a suitable value of m chosen as per wireless environment. This decreases the probability of missing I and hence caching of record of interest. If m is small then power dissipated by device is less. If m decreases, the chances that the cache be missed go up & the data access latency increases. The value of m therefore needs to be optimized which can be done by employing an appropriate algorithm.

Distributed Index-based Method

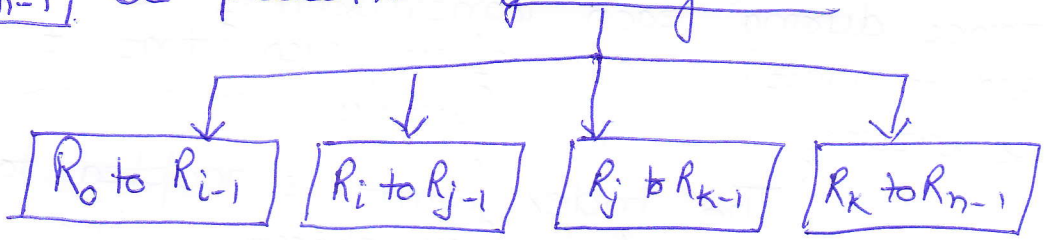
Distributed index-based method is an improvement on the (1,m) method. In this method, there is no need to repeat the complete index again and again. Instead of replicating the whole index m times, each index segment in a bucket only describle offset I' of data items. Each index I is partitioned into two parts I' and I''. I'' consists of unrepeated k levels, which do not repeat and I' consists of j repeated levels.

There are three distributed index methods:-

⇒ Non replicated distributed indexing
⇒ Fully replicated distributed indexing
⇒ Partially replicated distributed indexing.

Flexible Indexing Method.

Assume that a broadcast cycle has number of data segments with each of the segments having a variable set of records. For example, let n records, $\boxed{R_0 \text{ to } R_{n-1}}$ be present in four segments -



The possible index parameters are:-

⇒ I_{seg} having just 2 bits for offset, to specify location of segment

⇒ I_{rec} having just 6 bits of offset, to specify the location of a record of interest within segment.

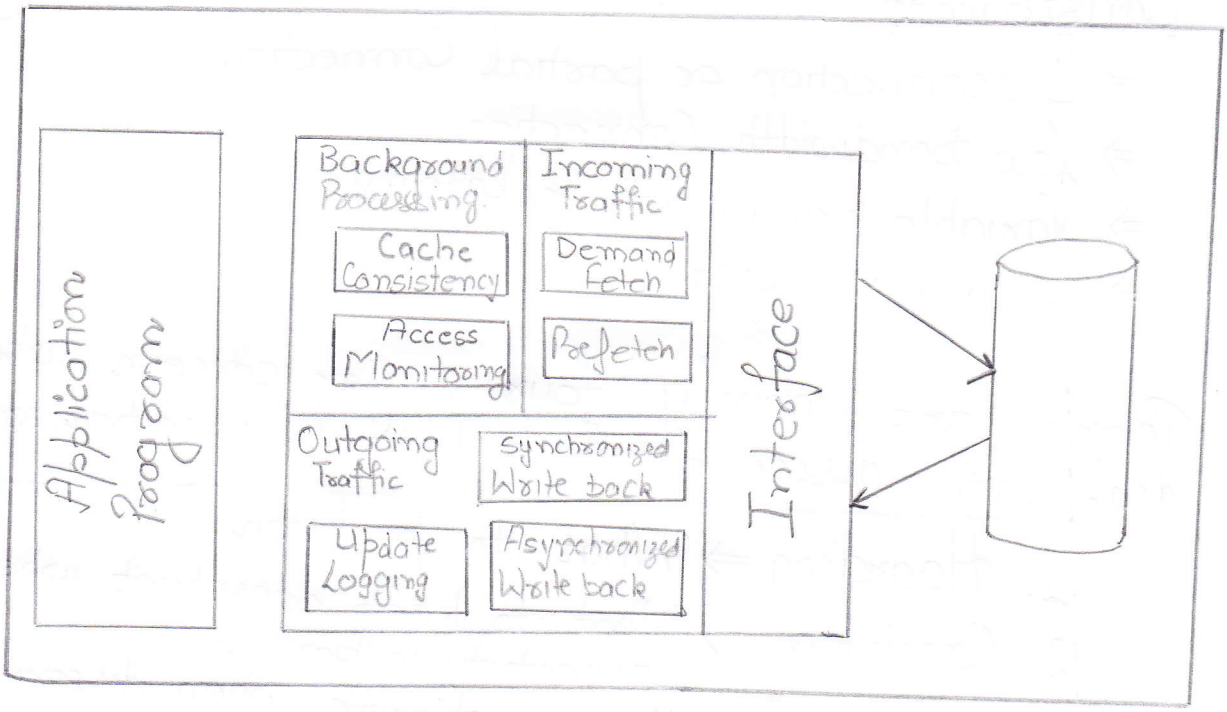
⇒ I_b having 4 bits for offset, to specify the location of a bucket of interest within record.

———————X———————

Mobile File System

UNIT-2nd

A mobile file system is a file system that is designed for the mobile environment. It is required to be simple and low on resources and should be extensible. The file system must be capable of sharing data over the wireless networks. The most important part of a Mobile File System is the cache manager, which intercepts file system operations from application programs and resolves them. The cache manager has a number of components that implement the cache management policy for file system in an effective manner. Following figure shows the simple architecture used for mobile file system.



The simplest architecture used as mobile file system mainly contains three modules such as incoming traffic, outgoing traffic and background processing.

Requirements for MFS

⇒ Access the same file as if connected.
⇒ Retain the same consistency when again connected.
⇒ Availability and Reliability as if connected.
⇒ Must posses ACID properties

A	→ Atomicity.
C	→ Consistent
I	→ Isolation
D	→ Durable

Constraints

⇒ Disconnection or partial Connection
⇒ Low bandwidth Connection
⇒ Variable bandwidth & latency.
⇒ Connection cost

There are mainly four major concern when deal with disconnection or partial connection problem.

1. Hoarding ⇒ What to prefetch
2. Consistency ⇒ What to keep consistent when connectivity is partial.
3. Emulation ⇒ How to operate when disconnected
4. Conflict Resolution ⇒ How to resolve conflicts.

Mainly two type of file systems are used for mobile environment. Standard file systems like NFS are very inefficient and almost unusable in a mobile & wireless environment. They do not expect disconnection, low bandwidth connections and high latencies. To support disconnected operation, portable device may replicate files or single objects. This can be done by prefetching or caching. But the main problem is to provide consistency with original data.

Distributed File System

DFS is a set of client and server services that allow a large enterprise to organize many distributed servers that are capable of sharing. DFS provides location transparency and redundancy to improve the data availability in case of failure or heavy load by allowing shares in multiple different locations to be logically grouped under one folder known as DFS ROOT.

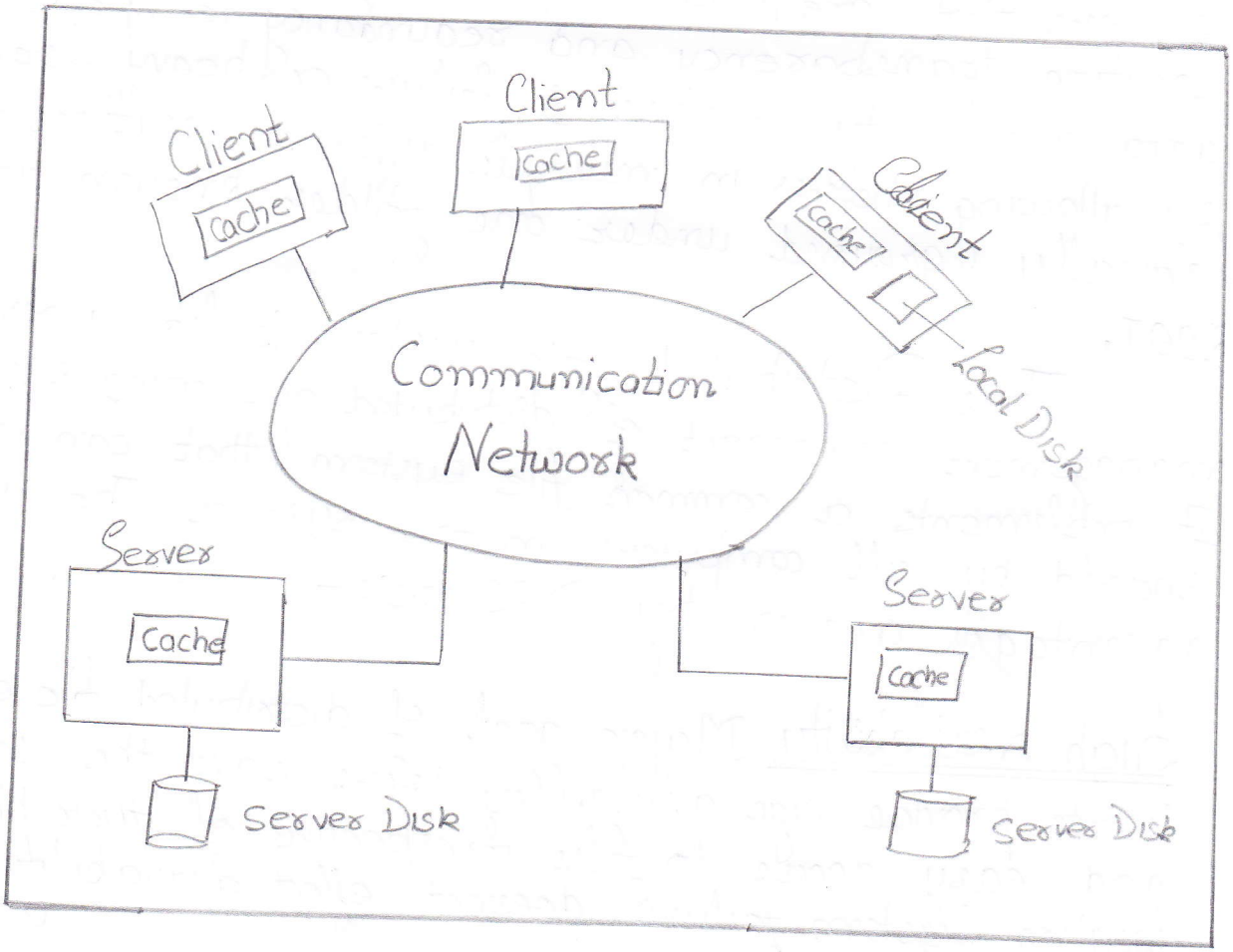
Thus, Distributed file system is like a resource management component of distributed operating system. It implements a common file system that can be shared by all computers in the system. The main advantages drawn by DFS are:-

High Availability Major goal of distributed file system is to provide high availability. Users have the same and easy access to files irrespective of their physical location. System failures doesnot effect availability of files.

Network Architecture Another goal of a distributed file system is to provide same functional capabilities to access files distributed over a network as file system works at one location.

Architecture of DFS

In a distributed file system, file can be stored at any machine and the computation can be performed at any machine. When a machine needs to access a file stored on a remote machine, the remote machine performs the necessary file access operations and returns data if a read operation is performed. For higher performance, several machines or collection of machines known as file servers are dedicated to storing files and performing storage and retrieval operations. The rest of machines in the system can be used for computational purposes. These machines are referred as Clients & they access the files stored on servers.



Most important functionalities provided in DFS are Name Server & Cache Manager which are described below :-

Name Server

A name server is a process that maps name specified by clients to stored objects such as files & directories. This mapping occurs when a process refrences a file or directory for the first time.

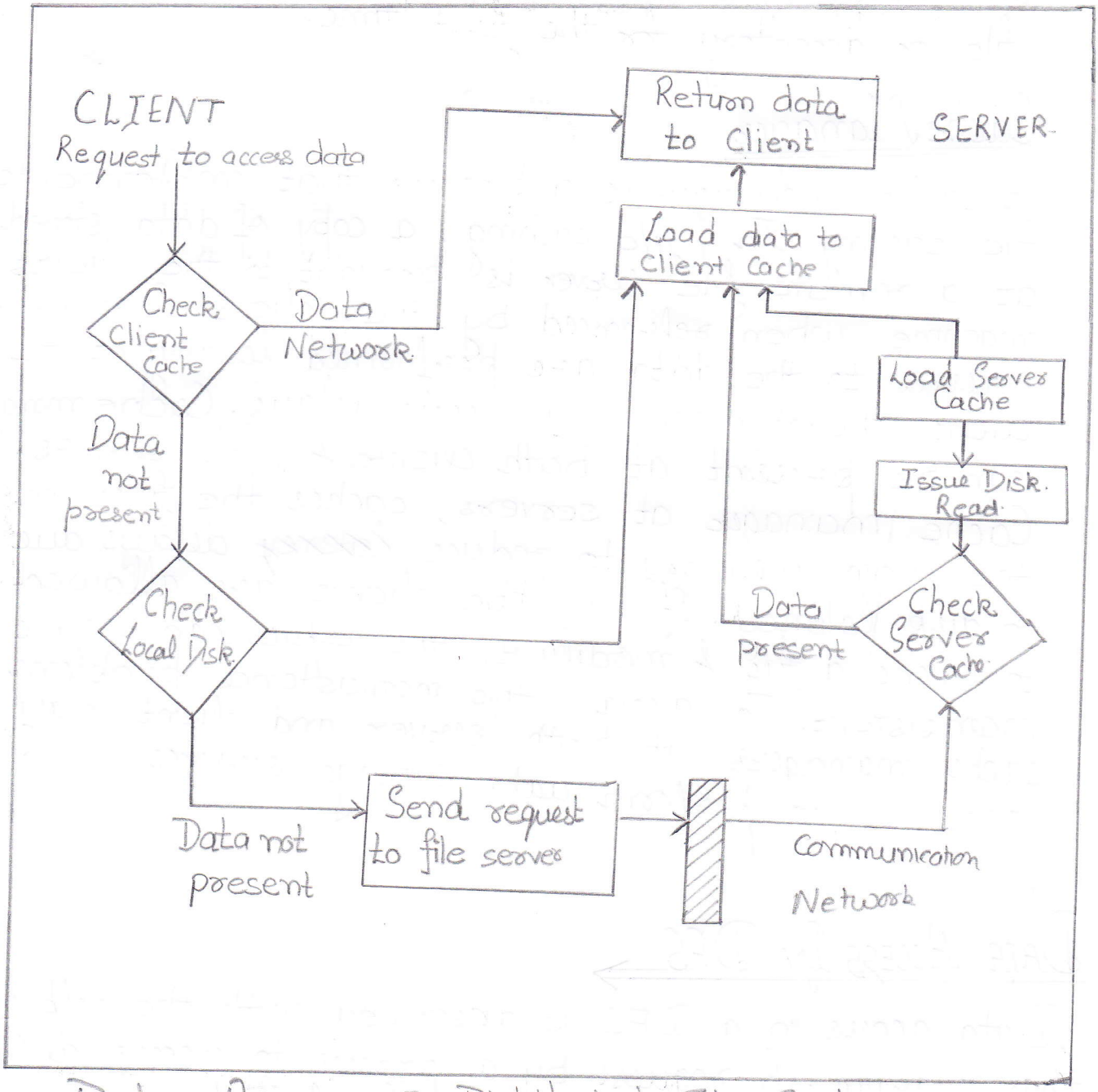
Cache Manager

A cache manager is a process that implements file caching. In file caching, a copy of data stored at a remote file server is brought to the client's machine when refrenced by the client. Subsequent accesses to the data are performed locally at the client, thereby reducing access delays. Cache manager can be present at both clients & file servers. Cache manager at servers, caches the files in the main memory to reduce ~~latency~~ delays due to disk latency. If multiple clients are allowed to cache a file & modify it, the copies can become inconsistent. To avoid this inconsistence problem, cache managers at both server and client may coordinate to perform data storage services.

DATA ACCESS IN DFS

Data access in a DFS is described with the help of flow diagram. A request by a process to access a data is first checked from the local cache of the machine on which the process is running. If the block is not in the cache, then the local disk, if present, is checked for the presence of the data block. If data block is present, then the request is satisfied and the

block is loaded into the client cache. If the block is not stored locally, then the request is passed on to the appropriate file server. The server checks its own cache for the presence of the data before issuing a disk I/O request. The data block is transferred to client cache in any case and loaded to the server cache if it was missing in the server cache.



Data Access In Distributed File System.

Design Issues Related to Distributed File System

1) Naming and Name Resolution

A name in file systems is associated with an object. Name resolution refers to the process of mapping a name to an object or multiple objects. Mainly three approaches are used to name files in a distributed environments.

a) Simplest scheme is to ~~concentrate~~ concatenate the host name to the names of files that are stored on that host. This approach guarantees that a filename is unique systemwide. Main problem with this approach is that moving a file from one host to another requires changes in the filename. That is, this naming scheme is not location independent.

b) Another approach is to mount remote directories onto local directories.

c) The third approach is to have a single global directory where all files in the system belong to a single name space.

2) Caches on Disk or Main Memory

This issue concerned with the question of whether the data cached by a client ~~sto~~ stored in the main memory at the client or local disk at the client.

Advantages of having cache in Main Memory

⇒ Diskless workstations are cheaper.

⇒ Accessing a cache from main memory is faster as compared to local disk.

⇒ A single design is used for a caching mechanism to both clients & servers.

Advantages of Caching on Local Disk.

⇒ Large files can be cached without affecting performance.

⇒ Virtual memory management is simple.

3) Writing Policy

To choose proper writing policy is another important concern for DFS. Writing policy is needed when a modified cache block at a client should be transferred to the server. The simplest policy is write-through. In write through, all writes requested by applications at clients are carried out at servers immediately. Main advantage of this policy is reliability. But in the case of crash, information is lost.

Delayed Writing Policy

This policy delays writing at the server. In this case, modifications due to a write are reflected at the server after some delay. This approach take the advantage of cache. Another variation of delayed writing policy is that it delays the updating of files at the server until the file is closed at the client.

4) Cache Consistency

To maintain consistency at both sides (client & server), several ways are used.

a) ⇒ In server-initiated approach, server informs cache managers whenever the data in the client caches changes.

b) ⇒ In client-initiated approach, client informs cache managers at the clients to validate data with the server before returning it to the clients.

5) Availability

Availability is one of the important issues in design of distributed file systems. Failure of servers or communication network can affect the availability of files. Replication is the primary mechanism used for enhancing the availability of files in distributed file system. Replication is the process of maintaining many copies or replicas of files maintained at different servers. Replication is expensive because of extra storage space required to store ~~space~~ the replicas & maintain the replicas upto date. The most two serious problem related to the replication are :—

⇒ How to keep the replica of a file consistent

⇒ How to detect inconsistencies among replicas of a file.

6) Scalability

The issue of scalability deals with the suitability of the design of a system to fulfill the demands of a growing system.

Advantages of using DFS

⇒ Files are more widely available since many computers can access the servers.

⇒ Sharing of the files is easy.

⇒ Servers can provide large storage space.

CODA

Coda is a distributed file system developed as a research project at CMU [Carneige Mellon University] since 1987 under the direction of Mahadev Satyanarayanan. Coda is distributed file system with its origin in AFS. Coda have several features :—

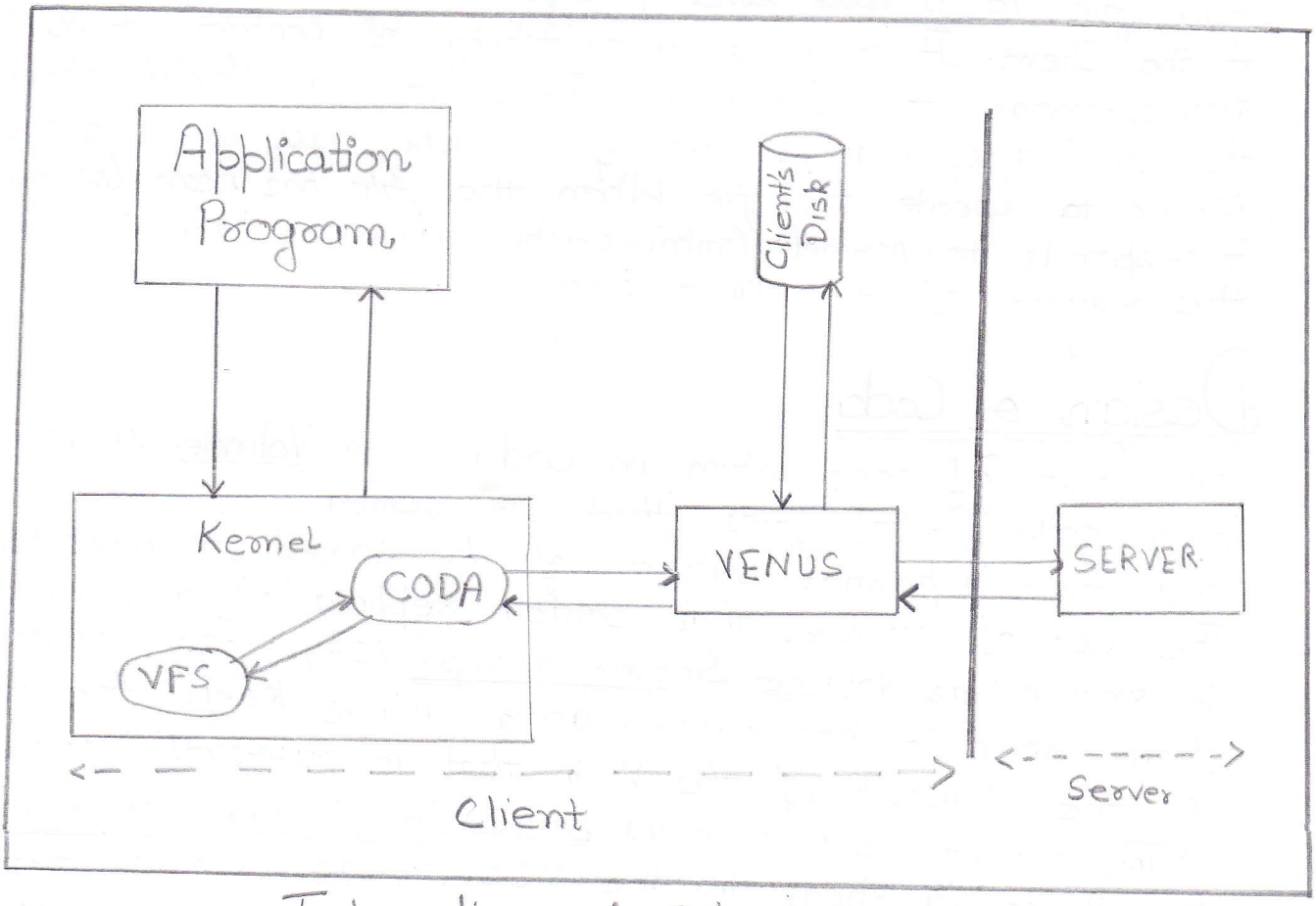
1. freely available source code under liberal license.
2. high performance
3. server replication.
4. continued operation during partial network failure.
5. network bandwidth adaptation.
6. well defined semantics of sharing even in the case of failure.
7. disconnected operations provides the reintegration of data from disconnected clients.
8. Failure resilience
9. Performance & Scalability.
10. Proper security model for authentication & encryption.

Why CODA ?

There are many problems faced by distributed file system. Transporting the files over the net can create poor performance, network bottleneck and server overload can result. The security of data is another important issue, i.e, how can we be sure that a client is really authorized to have access to information and how can we prevent the data. Failure is another main problem generally faced by DFS.

Coda was initially designed to address or solve the mentioned issues of DFS. Coda implemented as a prototype of highly available distributed file system for disconnected operation of clients. The disconnected mode operation supported by Coda file system is actually an initial step towards sharing and collaborative computation in a mobile computing environment.

Coda provides a common name space for all the files that clients share. Coda running on a client shows a file system of type coda mounted under /coda. All files provided by any of the servers are available to a client under the coda directory. Coda system running in a client will fetch the required file from the appropriate server and make it available to the client. The interactions of various Coda are depicted by following figure.



Interaction of Coda Components

Suppose a user wants to display the contents of a file in /coda directory, eg. by issuing the cat command. The user may be connected to network or operating in disconnected mode. The cat command generates a sequence of system calls like open, read and then write. All system calls are executed in kernel mode. The kernel at a client requires the services of a subsystem called VFS [Virtual File System] to open a file before it can service commands. VFS recognizes that the file arguments to be a Coda resource. It then informs the kernel module for Coda file system. This module is basically an in-kernel mini cache. It keeps a cache of recently answered requests for Coda files from VFS. If the file is locally available as container file within the cache area in local Coda file system, then VFS can service the call in kernel-mode.

In the case the file is not locally available, the mini-cache passes the request for the Coda file to a user level program called Venus running at the Client. It requires a switching of context from kernel-mode to user-mode. The client side Venus checks the local disk, and in case of a cache miss contacts the server to locate the file. When the file has been located it responds to kernel (mini-cache) which in turn services the request of the client program.

Design of Coda

The unit of replication in Coda is a Volume. A volume is a collection of files that are stored on one server and form a partial subtree of the shared file namespace. The set of servers that contain replicas of a volume is known as Volume Server Group (VSG). For each volume, Venus keeps the track from which it has cached data. of the subset of the VSG that is currently accessible. This subset is known as Accessible Volume Server Group and it is identical to the VSG in absence of failure. A server is deleted from AVSG when an operation on it times out. It is added back to the AVSG when VENUS is able to reestablish communication

Replication in CODA

CODA system use a variation of read-one, write all approach. When a file is closed after modification, it is transferred to all the members of AVSG. This approach is simple to implement and maximizes the probability that every replication site has current data at all times. Server CPU load is minimized because burden of data propagation is on the client rather than the server. This approach improves scalability. When servicing a cache miss, a client obtains data from one member of its AVSG called preferred server. The preferred server can be choosen at random or on the basis of performance criteria. Data is transferred only from one server, the other servers are contacted to verify that the preferred server have latest copy of the data.

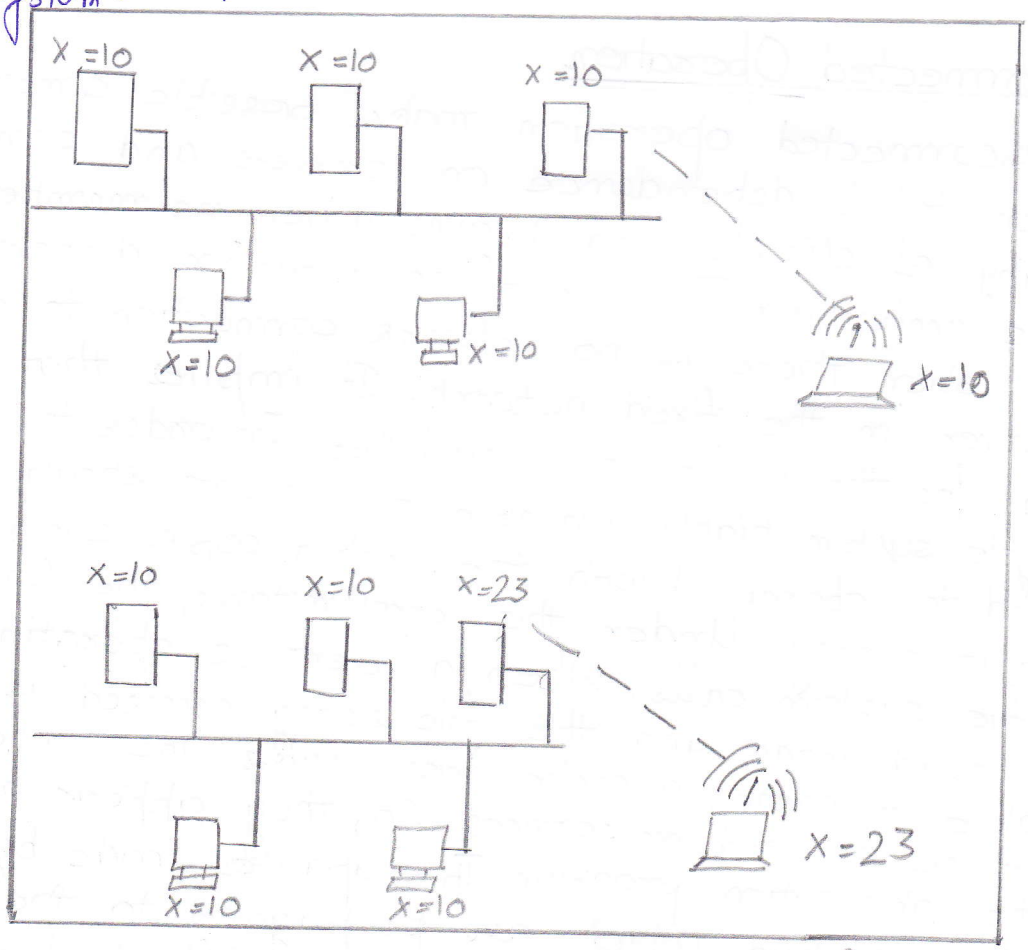
Disconnected Operation

Disconnected operation makes possible compromise between total dependence on servers and complete autonomy of clients and begins when no member of a VSG is accessible. The client operates in disconnected mode when there is no network connection to any site or server in the fixed network. It implies that the AVSG for the client is a null set. In order to make the file system highly available, the clients should be allowed to operate with the cached copies even when AVSG is empty. Under this circumstance, the Venus Services the file system calls. When a client is operating in disconnected mode and the file being accessed is not available in cache, a cache miss occurs. The cache misses cannot be masked or serviced. So, they appear as failures to the application program. The updates made by a client on the cached copies are propagated to the server when the client is reconnected. After propagation of the updates, Venus re-charges cache with the server replication.

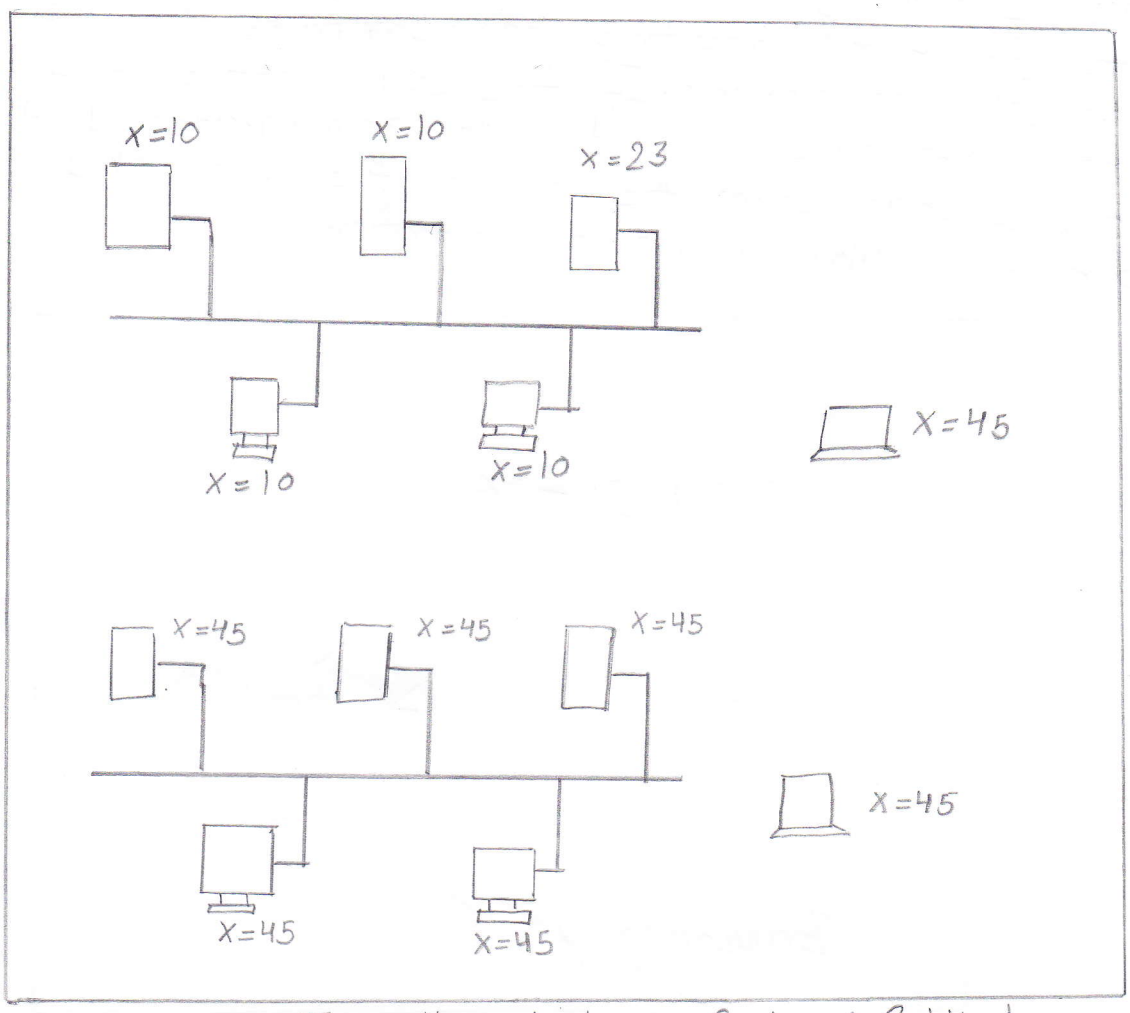
Transition b/w Client cache & replication at Client

This transition can be explained with the help of diagram. Figure a shows that all the clients including the mobile client are connected to the network. Therefore, the value of some object, say x, is identical to the network held by servers and the clients when all are connected. Figure (b) shows that a client operating in disconnected mode can update the value of the object in its cache, but this update will not be visible to server or to other clients even inside the same network partition. But when the disconnected client reconnects, the value of the object in it or at other nodes becomes same.

Coda project make distributed file system resilient to temporary network outages, crashes and failures and provide a highly available location-transparent distributed file system.



Transition between Cache & Replication



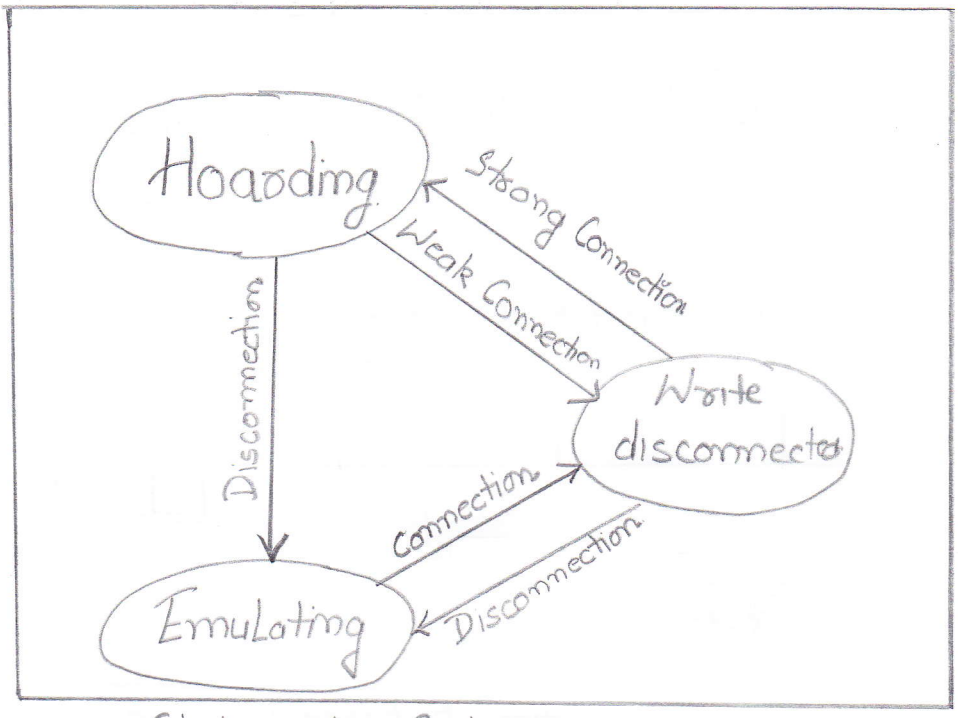
Transition between Cache & Replication

How CODA supports Mobility

To provide all necessary files for disconnected mode, Coda offers extensive mechanism for pre-fetching of file while still connected called hoarding. If the client is ~~+~~ connected to the server with a strong connection, hoarding transparently prefetches files currently used. A user can pre-determine a list of files, which Coda should explicitly pre-fetch. Additionaly, a user can assign priorities to certain programs. Coda now decides on the current cache content using the last and Least-recently used [LRU] strategy.

As soon as client is disconnected, applications work on the replicates & goes to emulating state. Coda follows an optimistic approach and allow read and write access to all files. The system keeps a record.

of changed files, but does not maintain a history of changes of each file. The cache always has only one replicate. After reconnection, Coda compares the replicates with the files on server. If the Coda notice that two different users have changed a file, re-integration of the file fails & Coda saves the changed file as a copy on the server to allow manual reintegration.



States in Coda.

The Client only performs hoarding while a strong connection to the server exists. If the connection breaks completely, the client goes into emulating and uses only the cached replicates. If the client loses the strong connection and only a weak connection remains, it does not perform hoarding, but decides if it should fetch the file in cache miss considering file type. The weak connection is not used for re-integration of files.

LITTLE WORK

The distributed file system Little work is like Coda, an extension of AFS. Little work only requires changes to the cache manager of the client and detects write conflicts during reintegration. Little Work has no specific tools for reintegration and offers no transaction service. Little Work uses more client states to maintain.

Little Work States

⇒ **Connected** The operation of the client is normal, i.e, no special mechanism are required. This mode needs a continous high bandwidth as available.

⇒ **Partially Connected**. If a client has only a lower bandwidth connection but still requires or have a possibility to communicate continously, it is referred to as partially connected. These networks typically based on the traffic, not based on the amount or duration of the connection.

Fetch Only If the only network available offers connections on demand, the client goes into fetch only state. The Client uses the replicates in the cache in an optimistic way, but fetches files via the communicaton link if they are not available in the cache. This enables a user to access all files of the server, but also tries to minimize the communication.

Disconnected Without any network, the client is in disconnected state.

FICUS

Ficus is a distributed file system which is not based on a client/Server approach. Ficus allows the optimistic use of replicates, detects write conflicts and solve conflicts on directories. Ficus uses so-called gossip protocols. A mobile computer does not necessarily need to have a direct connection to a server. With the help of other mobile computers, it can propagate updates through the network until it reaches a fixed network and the server. Thus, changes on files propagate through the network step by step. Ficus tries to minimize the exchanges of files that are valid only for short time, eg, temporary files. Main issue in this protocol is how fast they propagate to the client that needs this information & how much unnecessary traffic it causes to propagate information to clients that are not interested.

MIO-NFS

The system **M**obile **I**ntegration **O**f **NFS** is an extension of Network File System. In contrast to another file systems, MIO-NFS uses a pessimistic approach with tokens controlling access to files. Only the token-holder for a specific file may changes the file, so MIO-NFS avoid write conflicts. MIO-NFS support three modes.

- Connected
- Loosely Connected
- Disconnected.

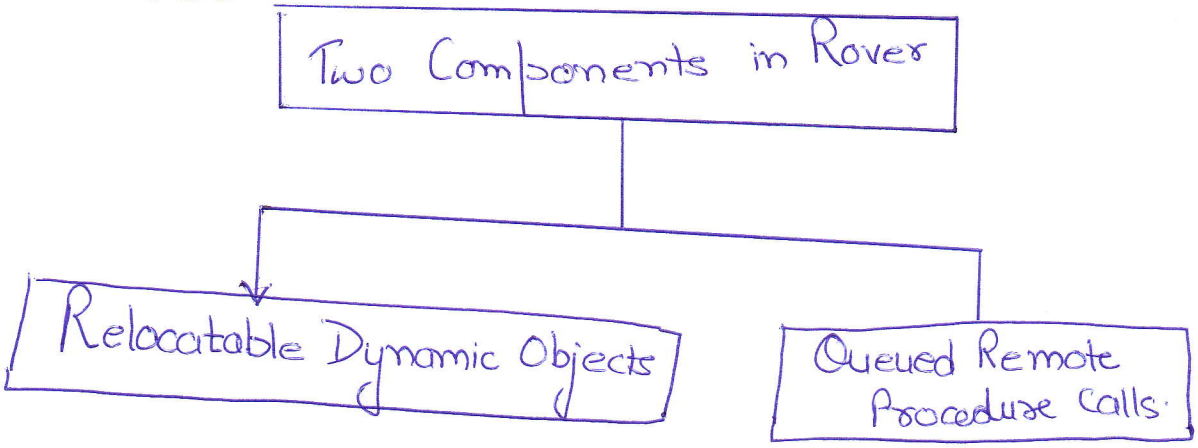
Connected Server handles all access of files as usual.

Loosely Connected Client uses local replicates, exchanges tokens over the networks and updates files via network.

Disconnected Client only uses local replicates. Writing is only allowed if client is token holder.

ROVER

Compared to Coda, the Rover platform uses another approach to support mobility. Instead of adapting existing applications for mobile devices. Rover provides a platform for developing new, mobility aware applications. Two new components are introduced in Rover.



Relocatable Dynamic Objects are those objects that can be dynamically loaded into the client computer from a server (or vice versa) to reduce client - server communication. A trade-off between transferring objects and transferring only data for objects has to be found. If a clients needs an object, it makes sense to migrate the object.

Queued Remote Procedure Calls allows for non-blocking RPCs even when a host is disconnected. Request & Responses are exchanged as soon as connection is available again.