Mobile Computing
Mr. Saurabh Singh

File System (Motivation):

The distributed filing system enables programs to accurately store and access remote files, and allows users to access files from any computer within the intranet.

Aims:

Efficiently and transparently access shared files during a mobile environment while maintaining data consistency.

Problem:

- Limited resources of mobile computers (memory, CPU, etc.)
- Low bandwidth, variable bandwidth, temporary disconnection
- High heterogeneity of hardware and software components (no standard PC architecture)
- Wireless network resources and mobile computers aren't very reliable.
- Standard file systems (such as NFS, network file systems) are very inefficient and almost unusable.

File system requirements:

1. Replication
2. Transparency (location, access, mobility, performance, zoom)
3. Current file update
4. Fault tolerance
5. Consistency
6. Security
7. Efficiency

Solutions:

Data replication (copy, clone, cache)
Collect data in advance (hoarding product, pre-fetching)

File systems - consistency problems:

The big problem of distributed, loosely coupled systems-

- Are all views of the data the same?
- How and when should the changes be propagated to whom?

Weak consistency-

- Many algorithms that provide strong consistency (for example, through atomic updates) cannot be used in a mobile environment
- If the mobile computer is not currently connected to the network, invalidating the data in the cache through the server is very problematic
- Sometimes inconsistencies must be tolerated, but then conflict resolution strategies must be applied to reach agreement again

Collision detection-

- Content irrelevant: version number, timestamp
- Content dependency: dependency graph

File systems for limited connectivity:

Symmetry-

- Client/server or peer relationship
- Support in fixed networks and/or mobile computers
- One file system or several file systems
- The namespace of a file or several namespaces

Transparency-

- Hide mobility support, applications on mobile computers should not notice mobility
- Users should not notice other mechanisms needed

Consistency model-

- Optimistic or pessimistic

Caching and prefetching-

- Single file, directory, sub trees, partition, etc.
- Permanent or only at certain points in time

File systems for limited connectivity II:

Data management-

- Manage buffered data and data copies
- Request for update, data validity
- Detect data changes

Conflict resolution-

- Application-specific or generic
- Error

There are several experimental systems-

- Coda (Carnegie Mellon University), Little work (University of Michigan), Ficus (UCLA), etc.
- Many systems use ideas from distributed file systems, such as AFS (Andrew File System)

CODA (Command Data Availability) File System:

Coda is a network file system developed as a research project of Carnegie Mellon University in 1987 under the guidance of Mahadev Satyanarayanan. It comes directly from the old version of AFS (AFS-2) and provides many similar functions. The Intermezzo file system was inspired by Coda. Coda is still under development, although the focus has shifted from research to creating powerful products that can be used for commercial purposes.

- Coda has many functions required by a network file system, as well as some functions that are not available elsewhere.
- Disconnect operation for mobile computing.
- Free under a free license.
- High performance through client-side persistent caching.
- Security model for authentication, encryption and access control.
- Continue to operate during part of the server network failure.
- Network bandwidth adaptation.
- Good scalability.
- Well-defined sharing semantics, even in the presence of network failures.
- Provide two different types of replication
    1. Server replication
    2. Cache on the client

Coda File System:

Disconnect operation of mobile client-

- Reintegrate data from disconnected clients
- Bandwidth adaptation

Resilience to failure-

- Read/write replication server
- Resolve server/server conflicts
- Handling of network failures of divided servers

- Handle customer disconnection

Performance and scalability-

- The client persistently caches files, directories and attributes for high performance
- Write-back cache

Safety-

- Kerberos like authentication
- Access control list (ACL)
- Well-defined shared semantics
- Free source code

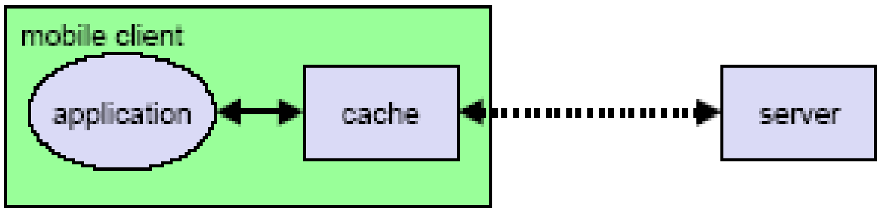
File systems - Coda I:

Transparent extension of client and server applications-

- Changes in the client's cache manager
- Cached copy of files used by applications
- Extensive and transparent data collection in advance for future use ("Hoarding product")

Consistency-

- The system keeps a record of file changes and compares files after reconnecting
- If different users change the same file, you need to manually reintegrate the file into the system
- Optimistic approach, coarse-grained (file size)



File systems - Coda II:

Hoarding:

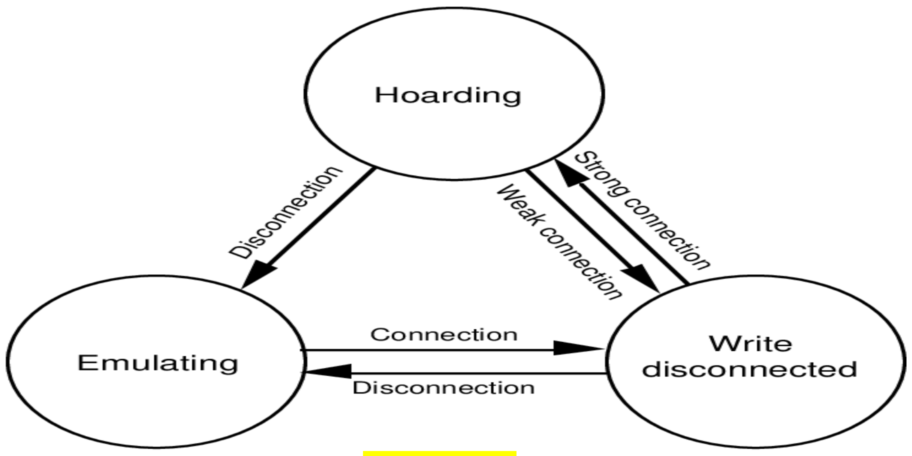
The function of size and bandwidth can hoard data whenever possible (if a network connection is available)-Hoarding
The user can pre-determine the priority file list
The content of the cache is determined by the list and LRU policy (least recently used)
Explicit prefetching possibilities
Regularly updated
Serving hoard/caching requests when disconnected-emulation
When the connection is regained, coordinate the repository/cache with the server and propagate the update – reintegrate/disconnect writing

File comparison:

Asynchronous, background
The system weighs update speed and minimizes network traffic

Cache miss:

User patient modelling: How long can the user wait for data without error messages?



Venus States
(Disconnected operation)

Hoarding:

Normal operation status (server is connected).
Venus will get useful data from the server to the client cache in preparation for disconnection.
Which data should be cached based on a priority algorithm, which consists of the most recent reference history and the storage database (HDB).
HDB is an entry for a path name that is used to identify objects that are explicitly stored (not cached) by the user.
When the cache is in a balanced state, Coda determines the priority of files placed in the cache.
Does the balance satisfy all these conditions?

- No un-cached files have higher priority than any cached files.
- The cache is full or there are no un-cached files with non-zero priority.

Emulation:

The state when disconnected.
Venus executes the server's jobs in the cache. It will record enough information in the log for replay during reintegration.
Each log contains system call parameters and the status of the objects referenced by the corresponding volume call.
If the client restarts, Venus also saves its cache and related data structures in non-volatile storage called Recoverable Virtual Memory (RVM).
RVM stores metadata for replay logs, HDB, cache directories, and status blocks.

Reintegration:

Restore the state of the server connection.
Venus updates the changes made in the emulation state to the server volume once.
Venus moves the replay log to ASVG in the server to be executed.
The replay algorithm is divided into 4 stages:

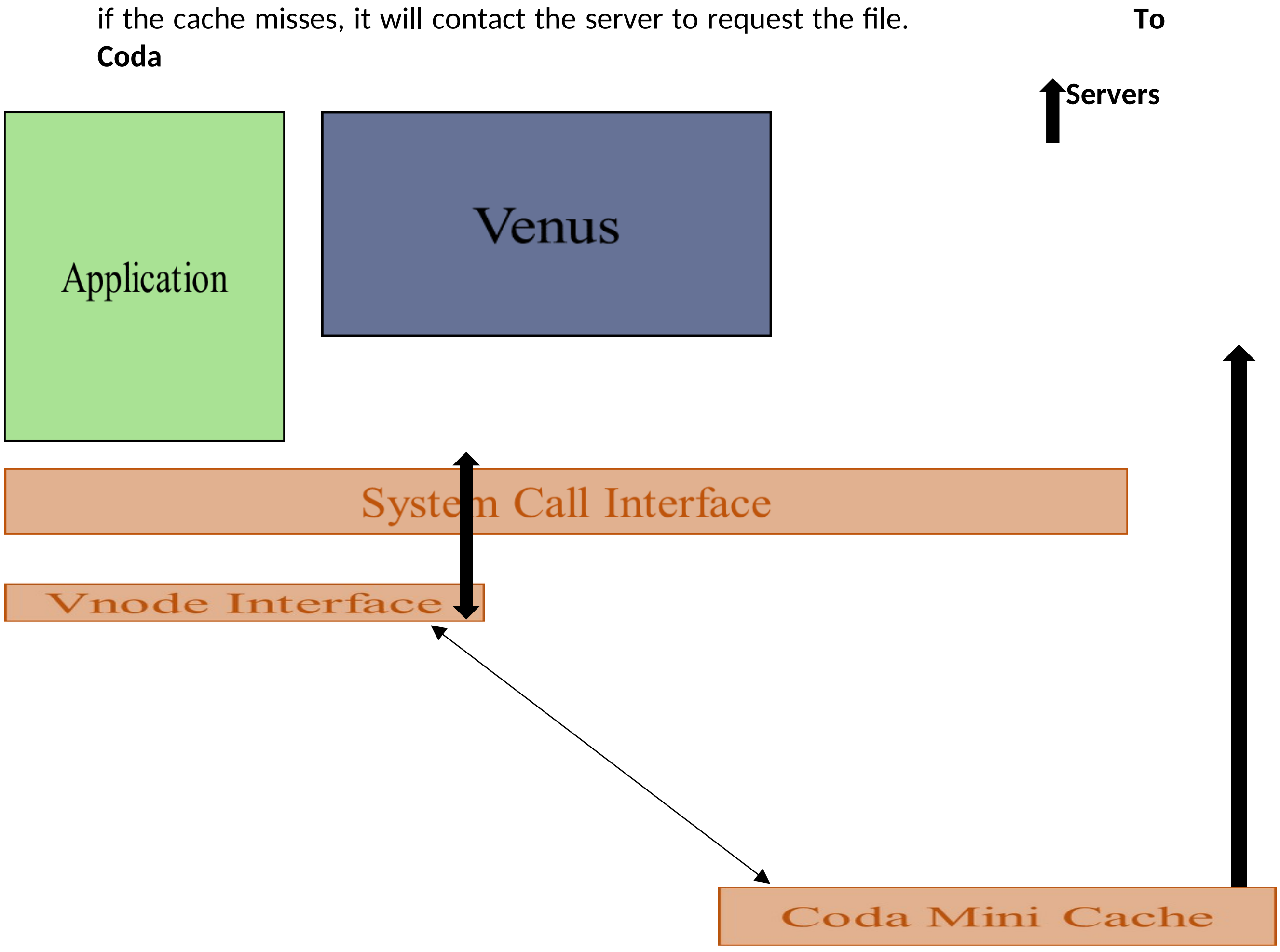
Phase 1: Parse the log and lock the reference object.
Phase 2: Verify the operation in the log.
Phase 3: Data transfer
Phase 4: Commit and release the lock.

Finally, Venus releases the replay log and resets the cache priority.

If there is a conflict (write/write) between the cache and the copy in the server, it will be resolved automatically or corrected manually according to the changed content.

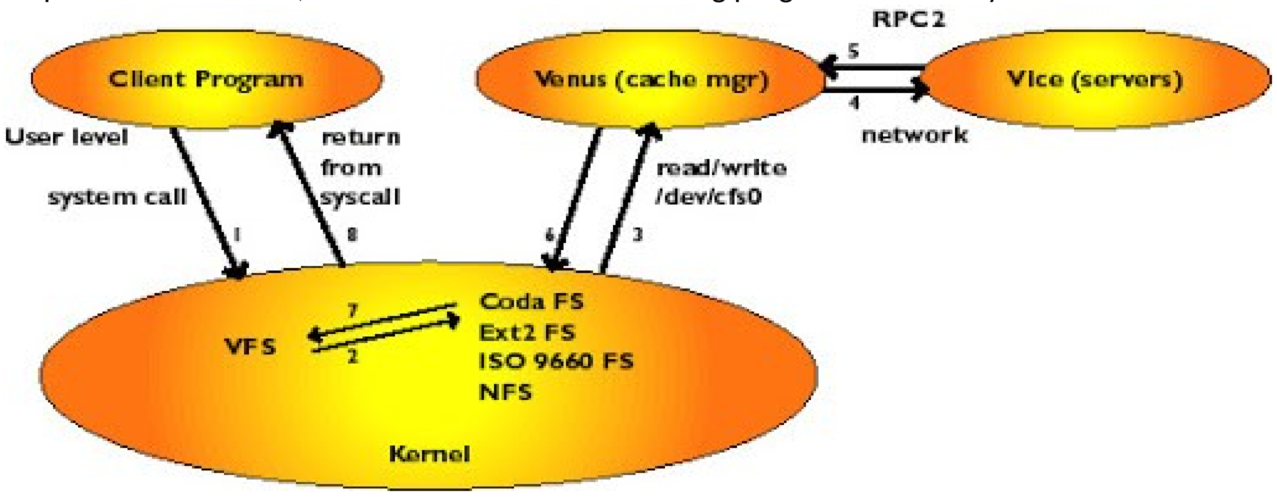
Structure of a Coda Client:

- The Coda client runs applications (such as a browser) or opens files on its system. Whenever the application generates a request, it will be forwarded to the file system through a system call. A system call is an operation in which a program requests a service from the kernel through this operation: for example, when a file is opened, the kernel will perform a search operation to find the index node of the file and return the file handle associated with the file to the kernel program. The index node contains the information used to access the data in the file and is used by the kernel, and the file handle is used to open the program. The system call is generated by the system call interface.
- In computing, the index node (index node) is a data structure on the traditional UNIX file system. An I node or v node (virtual file system node) stores all information about regular files, directories or other file system objects. Each file is associated with an index node, which is identified by an integer, usually called an i-node, which stores information about files and directories (folders), such as file ownership, access mode (read/write, execution permissions) and file type . The v node interface is used to provide an interface between the application and the kernel for system calls.
- The Coda client has a Coda mini cache, which stores a cache of the most recently responded requests by the VFS. This is all managed by the Cache Manager (Venus), which will check whether the required file is contained in the client's disk cache, and if the cache misses, it will contact the server to request the file.



How Coda Works:

In order to understand how Coda works after disconnecting from the server, let us analyze a simple file system operation. Suppose we type: ``cat / coda / tmp / foo'' to display the contents of the Coda file. What actually happened? The cat program will make some system calls for the file. A system call is an operation in which a program requests a service from the kernel through this operation: for example, when a file is opened, the kernel will perform a search operation to find the index node of the file and return the file handle associated with the file to the kernel. program. The index node contains the information used to access the data in the file and is used by the kernel, and the file handle is used to open the program. Open the call into the virtual file system in the kernel, and when you realize that the request is for a file in the /coda file system, the request will be handed over to the Coda file system module in the kernel. Coda is a very minimalistic file system module: it keeps a cache of recently responded requests from VFS, but otherwise passes the request to the Coda cache manager called Venus. Venus will check if there is tmp/foo in the client disk cache, and if the cache misses, it will contact the server to request tmp/foo. After finding the file, Venus responds to the kernel, which in turn returns the calling program from the system call.



When the kernel passes the open request to Venus for the first time, Venus uses a remote procedure call to reach the server, thereby fetching the entire file from the server. Then, it stores the file as a container file in the cache area (currently /usr/coda/venus.cache/). Now, the file is an ordinary file on the local disk, and Venus cannot be read and written to the file, but it is (almost) completely handled by the local file system (Linux ext2). The speed of Coda read and write operations is the same as that of local files. If you open the file a second time, it will not be retrieved from the server again, but the local copy can be used immediately. The catalog file (remember: a catalog is just a file) and all attributes (ownership, permissions, and size) are cached by Venus, and if the file exists in the cache, Venus allows the operation to continue without contacting the server. If the file has been modified and closed, Venus updates the server by sending a new file. Other operations that modify the file system (such as creating directories, deleting files or directories, and creating or deleting (symbolic links) are also propagated to the server.

File systems - Little Work:

- Only change the client's cache manager.
- Connection method and purpose.

	Connected	Partially Connected	Fetch Only	Disconnected
Method	**Normal**	**Delay write to the server**	**Optimistic Replication of files**	**Abort at cache miss**
Network Requirements	**Continuous High Bandwidth**	**Continuous Bandwidth**	**Connection on demand**	**None**
Application	**Office, WLAN**	**Packet Radio**	**Cellular Systems (e.g. GSM)with costs per call**	**Independent**

File systems - further examples:

Mazer/Tardo-

- File synchronization layer between application and native filing system.
- Cache complete subdirectories from the server.
- When necessary, the "redirector" responds to local requests, if possible, through the network.
- Regular consistency check with two-way update.

Ficus-

- Not a client/server method.
- Optimistic method based on replication, write conflict detection, conflict resolution.
- Use the "gossip" protocol: mobile computers do not necessarily need to be directly connected to the server, with the help of other mobile computers, updates can be spread over the network.

MIo-NFS (mobile integration of NFS)-

- NFS extension, pessimistic method, only token holders can write.
- Connected/loose/disconnected.

Disconnected operation:

The three states of Venus

Hoarding-

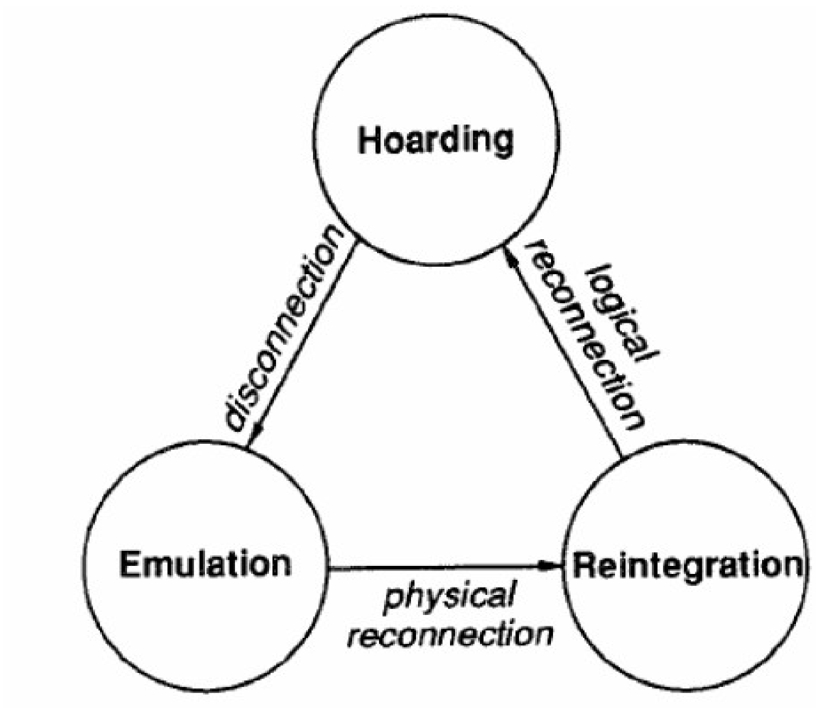
- Network connected.
- Collect important documents.

Emulation-

- Offline mode.
- Venus Simulation Server.

Reintegration-

- Network connected.
- Sync files.



Disconnected operation

How a user program asks for service from the kernel through a system call:

- The kernel passes the request to Venus by allowing Venus to read the request from the character device/dev/cfso. Venus attempts to answer the request by looking at its cache, asking the server or possibly by declaring disconnection to serve in disconnected mode.
- When there is no n/w connection with any server that owns the file, the disconnected mode will be started; similarly, if the server fails, the disconnected operation can also take effect.
- When the kernel passes the open request to the place for the first time, Venus uses RPC (Remote Procedure Call) to get the entire file from the server to reach the server. Then, it stores the file as a container file in the cache area.
- The file is now a normal file on the local disk, and Venus cannot be read and written to the file, but it is (almost) completely processed by the local file system. The speed of Coda read and write operations is the same as that of local files.
- If the file is opened for the second time, it will not be retrieved from the server again, but the local copy will be available immediately.
- If the file has been modified and closed, Venus updates the server by sending a new file.
- Therefore, Coda caches all the information it needs on the client and only notifies the server of updates to the file system.

From Caching to Disconnected Operation:

As we all know, coda caches all the information needed to access the data. When the file system is updated, these updates need to be propagated to the server. In normal connection mode, this type of update will be propagated to the server synchronously, that is, when the update is completed on the client, it has also been performed on the server. If the server is unavailable, such operations will fail with a timeout error. In this case, the error must be reported to the calling program. However, timeouts can be handled gracefully as follows:
In order to support disconnected computers or running with n/w failures, Venus will not report the failure to the user when the update causes a timeout. Instead, Venus realized that when the server was unavailable, the update should be recorded on the client. During disconnection, all updates are stored in the CML (Client Modification Log), which is frequently flushed to disk.

Disconnected operation :

- It only provides access to cached data.
- When the disconnected operation ends, the modified files and directories in the disconnected volume will be propagated to AVSG ().
- It is important to avoid cache loss during the disconnected operation.
- Coda allows users to specify a prioritized list of files and directories that Venus should strive to keep in the cache.
- The disconnection operation can also be entered voluntarily.
- Coda uses a version vector mechanism to detect inconsistencies.

Application of CODA:

- Network Server.
- Share files: for example, in a Beowulf cluster or across the enterprise.
- Especially where NFS is insufficient.
- Move/disconnect operation.
- Wan.

Database systems in mobile environments:

Request processing-

- Energy saving, location dependent, cost effective.
- Example: Find the fastest way to the hospital.

Copy/Replication management-

- Similar to file system.

Location management-

- Track mobile users to provide replicated or location-related data in the correct location in a timely manner (minimize access latency to the minimum)
- Example: With the help of HLR (Home Location Register) in GSM, mobile users can find local traction services

Transaction processing-

- "Mobile" transactions do not necessarily rely on the same model as transactions on fixed networks (ACID: atomicity, consistency, isolation, durability)
- So model "weak" transactions.

Mobile Code:

"Mobile code" is defined as "software that runs on a heterogeneous network, crosses the protection domain, and executes automatically after reaching the destination"
Protection domain-corporate network or PDA
The exclusion code is-

- Load from shared disk
- Download from the Internet (manually)

Mobile code supports a flexible form of distributed computing, where there is no need to know the required non-local computing in advance at the execution site
Advantages

- Effectiveness
- Simple and flexible
- Storage

Mobile code- A technique for transferring code from a computer system that stores code files to a computer system that executes the code. Java Applets are well-known examples of mobile codes. They are small programs in portable and interpretable byte code format and transferred from a web server to a web browser for execution as part of an HTML page.
Mobile agent- A special type of mobile code or program that can migrate from the initial host in a heterogeneous computer system network to many other hosts and complete tasks specified by its owner. In the self-initiated migration process, the agent will carry all its code and data, and in some systems will also carry a certain execution state.

Well-known examples of mobile code:

1. Postscript
2. Database Technology-SQL
3. Documents with embedded executable content transmitted on the network = Java (small program)
4. Inferno developed by Lucent-a network operating system with mobile code

As far as programming languages are concerned, the common requirements of mobile code

- Portability
- Safety
- Security
    - Confidentiality
    - Integrity
    - Availability
    - Authenticity
- Effectiveness

Representative programming language for mobile code

- Java
- Limbo
- Objective camel
- Obliq
- Telegraph
- Secure Tcl

Migration:

Process migration- Transfer the OS process from one m/c to another
The migration mechanism handles the binding between the process and the execution environment (for example open fds, env variables)
Provide load balancing
Provide transparent process migration
Provide some control
Such as external signal or migrate () system call
Object migration- objects can be moved between address spaces

Agent:

Program acts on behalf of the user
Enter the network on behalf of the user
Autonomous migration in the network
Perform calculations on behalf of the user
Proxy path

- Scheduled
- Dynamic agent determination

Reduce N/W usage
Increase the asynchrony between client and server
Add the client specified by the function to the server
Introducing concurrency
Agents usually have the following (or all) characteristics:
1. Autonomy
2. Adaptive/learning
3. Mobile
4. Perseverance or Persistent
5. Goal-oriented
6. Flexible
7. Communication/collaboration
8. Active/Proactive

Mobile Agent:

Mobile agents are defined as active objects (or clusters of objects) with behaviour, state, and location.
Mobility-Agents that can travel on the network
Autonomy- The agent decides when and where to move next

- Independent, identifiable computer programs and their code, data, and execution status can be moved in a heterogeneous network of computer systems.
- The execution can be suspended at any point and transferred to another computer system.
- During this migration, the agent is completely transferred, that is, as a set of code, data and execution state.
- At the target location, execution will resume at the exact location where it was suspended.
- The client can maintain its own interface on the server node-the mobile agent acts as a proxy

- A given task can be divided into multiple tasks and distributed among mobile agents – to achieve parallelism
- Can use mobile agent paradigm-
    a. For low-level system management
    b. For middleware user-level application

Stationary VS Mobile:

Fixed agent or Stationary Agent-

- Can sit there and communicate with its environment through conventional means (RMI and messaging)
- Execute only on the system that started execution

Mobile Agent-

- Travel freely between hosts in the network
- Shipping is status and code

Mobile Agent VS Java Applets:

MA initiates the migration process; the migration of the Java applet is initiated from other software components (such as a web browser)
Java applets are only migrated from server to client, not from one client to another or back to the server
When the browser terminates or requests another web page, the life cycle of the Applet will end with the life cycle of the web page. MA usually migrate more than once

The Mobile Agent Model:

- Utilize code mobility on the Internet scale-
    i. Large heterogeneous host, technology
    ii. Strong (allow the entire.... code to move) v/s weak mobility (only move the data state)
- Mobility is location-aware-
    i. Programming language
    ii. Provide mechanisms and abstractions to transfer code to nodes or extract code from nodes
- Basic runtime-
    i. Support marshalling (in computer programming, marshalling is the following process: collect data from one or more applications or non-continuous sources in computer storage, put data fragments into message buffers, and then organize or convert the data into regulations The format of the process is applicable to a specific receiver or programming interface), code, check-in, security, etc.
    ii. Don't understand immigration policy
- Application field-
    i. Load balancing
    ii. E-commerce, distributed information retrieval, workflow...
- Mobile agent receives customer request
- Mobile agent enters fixed network
- The mobile agent acts as a client of the server
- Mobile agent performs conversion and filtering
- After connecting to the client, the mobile agent returns to the mobile platform

The Mobile Agent Model: Design

Several examples:

- **Client server** (remote execution:-the agent has been transferred to a remote node before activation, and runs on that node until it terminates)
- **Remote evaluation** (transfer an operation (a process + parameters) to a remote site, and perform the operation completely here)
- **Coding on demand** (the destination itself initiates the transmission of the program code. (ActiveX, Java Applets))
- **Mobile agent** (can migrate between nodes in the network)

Example:

i. Two friends Ram and Ramesh
ii. Interact to learn pizza making (service result)
iii. Recipe required (knowledge about service)
iv. There are ingredients (removable resources)
v. Oven baking (difficult to move resources)
vi. One person mixes ingredients according to the recipe (computing component responsible for executing the code)
vii. Preparing the cake (performing service)
viii. The place where the cake is prepared (execution place)

Client-Server: CS

Ram wants to eat a Pizza:

- He doesn't know the recipe
- He has no necessary ingredients and no oven

Ram knows Ramesh (likes baking Pizza)

- Know the recipe
- Has a fully equipped kitchen

Ram called Ramesh to ask:

- "Can you make me a pizza?"

Ramesh makes pizza and delivers them to Ram

Remote Evaluation: REV

Ram wants to eat a pizza:

- He knows the recipe
- He has no necessary ingredients and no oven

Ram knows Ramesh (likes baking pizza)

- Has a fully equipped kitchen
- Don't know the recipe

Ram called Ramesh to ask:

- "Can you make me a pizza? This is the recipe..."

Ramesh makes pizza and delivers them to Ram

Code on Demand: COD

Ram wants to eat a pizza:

- He has the necessary ingredients and oven
- He doesn't know the recipe

Ram knows Ramesh

- Know the recipe (and willing to share)

Ram called Ramesh to ask:

- "Can you tell me the recipe for making pizza?"

Ramesh told him the recipe, Ram makes pizza

Mobile Agent: MA

Ram wants to eat a pizza:

- He has the recipe and the required ingredients
- He doesn't have an oven

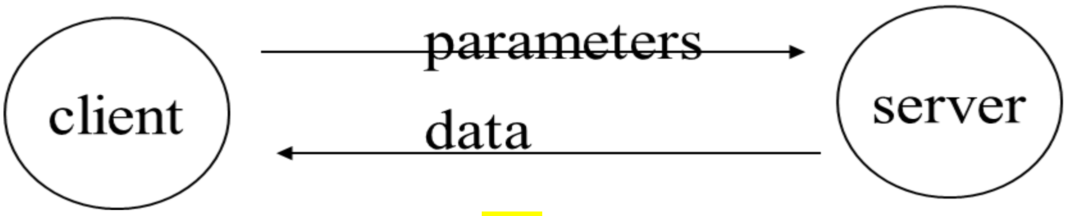
Ram knows Ramesh

- Have an oven (and willing to share)

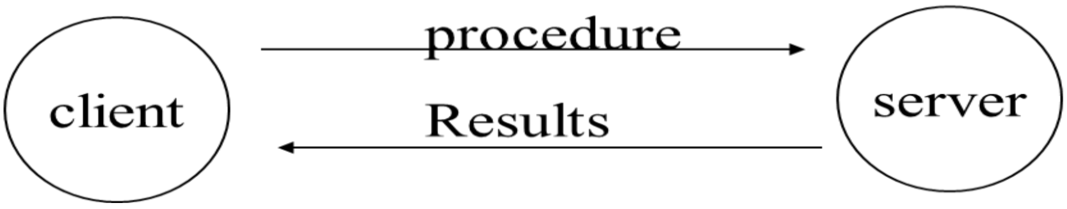
Ram can

- Prepare the batter
- Where to go to Ramesh
- Bake a pizza

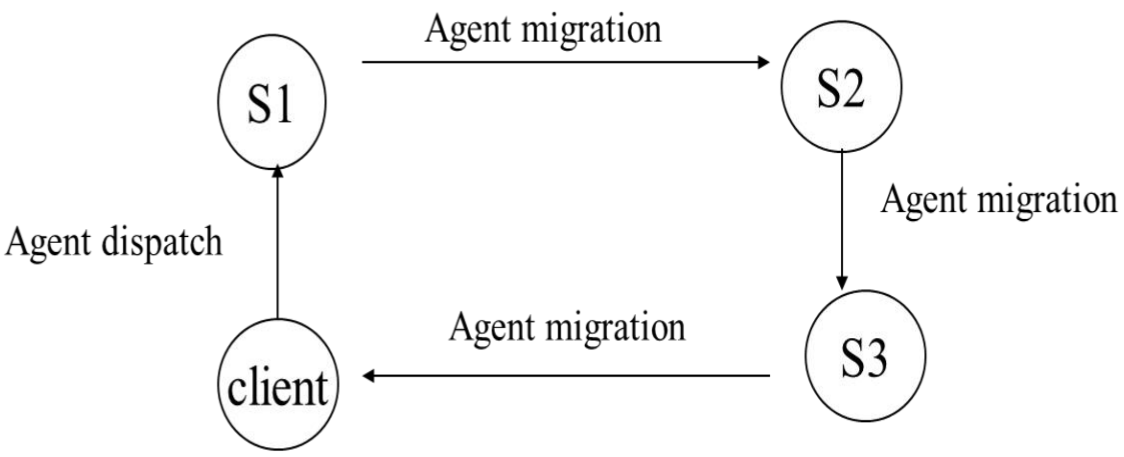
Several other changes are also possible



RPC



REV



Mobile Agent

File System (Motivation):

The distributed filing system enables programs to accurately store and access remote files, and allows users to access files from any computer within the intranet.

Aims:

Efficiently and transparently access shared files during a mobile environment while maintaining data consistency.

Problem:

- Limited resources of mobile computers (memory, CPU, etc.)
- Low bandwidth, variable bandwidth, temporary disconnection
- High heterogeneity of hardware and software components (no standard PC architecture)
- Wireless network resources and mobile computers aren't very reliable.
- Standard file systems (such as NFS, network file systems) are very inefficient and almost unusable.

File system requirements:

1. Replication
2. Transparency (location, access, mobility, performance, zoom)
3. Current file update
4. Fault tolerance
5. Consistency
6. Security
7. Efficiency

Solutions:

Data replication (copy, clone, cache)
Collect data in advance (hoarding product, pre-fetching)

File systems - consistency problems:

The big problem of distributed, loosely coupled systems-

- Are all views of the data the same?
- How and when should the changes be propagated to whom?

Weak consistency-

- Many algorithms that provide strong consistency (for example, through atomic updates) cannot be used in a mobile environment
- If the mobile computer is not currently connected to the network, invalidating the data in the cache through the server is very problematic
- Sometimes inconsistencies must be tolerated, but then conflict resolution strategies must be applied to reach agreement again

Collision detection-

- Content irrelevant: version number, timestamp
- Content dependency: dependency graph

File systems for limited connectivity:

Symmetry-

- Client/server or peer relationship
- Support in fixed networks and/or mobile computers
- One file system or several file systems
- The namespace of a file or several namespaces

Transparency-

- Hide mobility support, applications on mobile computers should not notice mobility
- Users should not notice other mechanisms needed

Consistency model-

- Optimistic or pessimistic

Caching and prefetching-

- Single file, directory, sub trees, partition, etc.
- Permanent or only at certain points in time

File systems for limited connectivity II:

Data management-

- Manage buffered data and data copies
- Request for update, data validity
- Detect data changes

Conflict resolution-

- Application-specific or generic
- Error

There are several experimental systems-

- Coda (Carnegie Mellon University), Little work (University of Michigan), Ficus (UCLA), etc.
- Many systems use ideas from distributed file systems, such as AFS (Andrew File System)

CODA (Command Data Availability) File System:

Coda is a network file system developed as a research project of Carnegie Mellon University in 1987 under the guidance of Mahadev Satyanarayanan. It comes directly from the old version of AFS (AFS-2) and provides many similar functions. The Intermezzo file system was inspired by Coda. Coda is still under development, although the focus has shifted from research to creating powerful products that can be used for commercial purposes.

- Coda has many functions required by a network file system, as well as some functions that are not available elsewhere.
- Disconnect operation for mobile computing.
- Free under a free license.
- High performance through client-side persistent caching.
- Security model for authentication, encryption and access control.
- Continue to operate during part of the server network failure.
- Network bandwidth adaptation.
- Good scalability.
- Well-defined sharing semantics, even in the presence of network failures.
- Provide two different types of replication
    1. Server replication
    2. Cache on the client

Coda File System:

Disconnect operation of mobile client-

- Reintegrate data from disconnected clients
- Bandwidth adaptation

Resilience to failure-

- Read/write replication server
- Resolve server/server conflicts
- Handling of network failures of divided servers

- Handle customer disconnection

Performance and scalability-

- The client persistently caches files, directories and attributes for high performance
- Write-back cache

Safety-

- Kerberos like authentication
- Access control list (ACL)
- Well-defined shared semantics
- Free source code

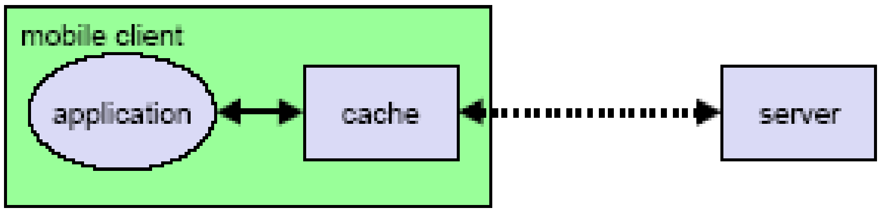
File systems - Coda I:

Transparent extension of client and server applications-

- Changes in the client's cache manager
- Cached copy of files used by applications
- Extensive and transparent data collection in advance for future use ("Hoarding product")

Consistency-

- The system keeps a record of file changes and compares files after reconnecting
- If different users change the same file, you need to manually reintegrate the file into the system
- Optimistic approach, coarse-grained (file size)



File systems - Coda II:

Hoarding:

The function of size and bandwidth can hoard data whenever possible (if a network connection is available)-Hoarding

The user can pre-determine the priority file list

The content of the cache is determined by the list and LRU policy (least recently used)

Explicit prefetching possibilities

Regularly updated

Serving hoard/caching requests when disconnected-emulation

When the connection is regained, coordinate the repository/cache with the server and propagate the update – reintegrate/disconnect writing

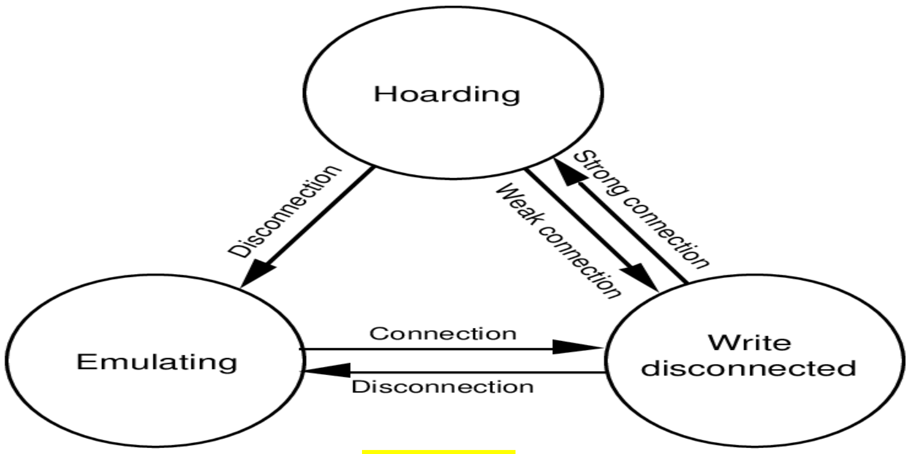
File comparison:

Asynchronous, background

The system weighs update speed and minimizes network traffic

Cache miss:

User patient modelling: How long can the user wait for data without error messages?



Venus States
(Disconnected operation)

Hoarding:

Normal operation status (server is connected).
Venus will get useful data from the server to the client cache in preparation for disconnection.
Which data should be cached based on a priority algorithm, which consists of the most recent reference history and the storage database (HDB).
HDB is an entry for a path name that is used to identify objects that are explicitly stored (not cached) by the user.
When the cache is in a balanced state, Coda determines the priority of files placed in the cache.
Does the balance satisfy all these conditions?

- No un-cached files have higher priority than any cached files.
- The cache is full or there are no un-cached files with non-zero priority.

Emulation:

The state when disconnected.
Venus executes the server's jobs in the cache. It will record enough information in the log for replay during reintegration.
Each log contains system call parameters and the status of the objects referenced by the corresponding volume call.
If the client restarts, Venus also saves its cache and related data structures in non-volatile storage called Recoverable Virtual Memory (RVM).
RVM stores metadata for replay logs, HDB, cache directories, and status blocks.

Reintegration:

Restore the state of the server connection.
Venus updates the changes made in the emulation state to the server volume once.
Venus moves the replay log to ASVG in the server to be executed.
The replay algorithm is divided into 4 stages:
Phase 1: Parse the log and lock the reference object.
Phase 2: Verify the operation in the log.
Phase 3: Data transfer
Phase 4: Commit and release the lock.
Finally, Venus releases the replay log and resets the cache priority.

If there is a conflict (write/write) between the cache and the copy in the server, it will be resolved automatically or corrected manually according to the changed content.

File System (Motivation):

The distributed filing system enables programs to accurately store and access remote files, and allows users to access files from any computer within the intranet.

Aims:

Efficiently and transparently access shared files during a mobile environment while maintaining data consistency.

Problem:

- Limited resources of mobile computers (memory, CPU, etc.)
- Low bandwidth, variable bandwidth, temporary disconnection
- High heterogeneity of hardware and software components (no standard PC architecture)
- Wireless network resources and mobile computers aren't very reliable.
- Standard file systems (such as NFS, network file systems) are very inefficient and almost unusable.

File system requirements:

1. Replication
2. Transparency (location, access, mobility, performance, zoom)
3. Current file update
4. Fault tolerance
5. Consistency
6. Security
7. Efficiency

Solutions:

Data replication (copy, clone, cache)
Collect data in advance (hoarding product, pre-fetching)

File systems - consistency problems:

The big problem of distributed, loosely coupled systems-

- Are all views of the data the same?
- How and when should the changes be propagated to whom?

Weak consistency-

- Many algorithms that provide strong consistency (for example, through atomic updates) cannot be used in a mobile environment
- If the mobile computer is not currently connected to the network, invalidating the data in the cache through the server is very problematic
- Sometimes inconsistencies must be tolerated, but then conflict resolution strategies must be applied to reach agreement again

Collision detection-

- Content irrelevant: version number, timestamp
- Content dependency: dependency graph

File systems for limited connectivity:

Symmetry-

- Client/server or peer relationship
- Support in fixed networks and/or mobile computers
- One file system or several file systems
- The namespace of a file or several namespaces

Transparency-

- Hide mobility support, applications on mobile computers should not notice mobility
- Users should not notice other mechanisms needed

Consistency model-

- Optimistic or pessimistic

Caching and prefetching-

- Single file, directory, sub trees, partition, etc.
- Permanent or only at certain points in time

File systems for limited connectivity II:

Data management-

- Manage buffered data and data copies
- Request for update, data validity
- Detect data changes

Conflict resolution-

- Application-specific or generic
- Error

There are several experimental systems-

- Coda (Carnegie Mellon University), Little work (University of Michigan), Ficus (UCLA), etc.
- Many systems use ideas from distributed file systems, such as AFS (Andrew File System)

CODA (Command Data Availability) File System:

Coda is a network file system developed as a research project of Carnegie Mellon University in 1987 under the guidance of Mahadev Satyanarayanan. It comes directly from the old version of AFS (AFS-2) and provides many similar functions. The Intermezzo file system was inspired by Coda. Coda is still under development, although the focus has shifted from research to creating powerful products that can be used for commercial purposes.

- Coda has many functions required by a network file system, as well as some functions that are not available elsewhere.
- Disconnect operation for mobile computing.
- Free under a free license.
- High performance through client-side persistent caching.
- Security model for authentication, encryption and access control.
- Continue to operate during part of the server network failure.
- Network bandwidth adaptation.
- Good scalability.
- Well-defined sharing semantics, even in the presence of network failures.
- Provide two different types of replication
    1. Server replication
    2. Cache on the client

Coda File System:

Disconnect operation of mobile client-

- Reintegrate data from disconnected clients
- Bandwidth adaptation

Resilience to failure-

- Read/write replication server
- Resolve server/server conflicts
- Handling of network failures of divided servers

- Handle customer disconnection

Performance and scalability-

- The client persistently caches files, directories and attributes for high performance
- Write-back cache

Safety-

- Kerberos like authentication
- Access control list (ACL)
- Well-defined shared semantics
- Free source code

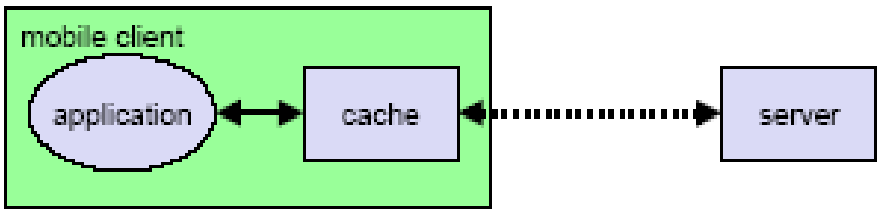
File systems - Coda I:

Transparent extension of client and server applications-

- Changes in the client's cache manager
- Cached copy of files used by applications
- Extensive and transparent data collection in advance for future use ("Hoarding product")

Consistency-

- The system keeps a record of file changes and compares files after reconnecting
- If different users change the same file, you need to manually reintegrate the file into the system
- Optimistic approach, coarse-grained (file size)



File systems - Coda II:

Hoarding:

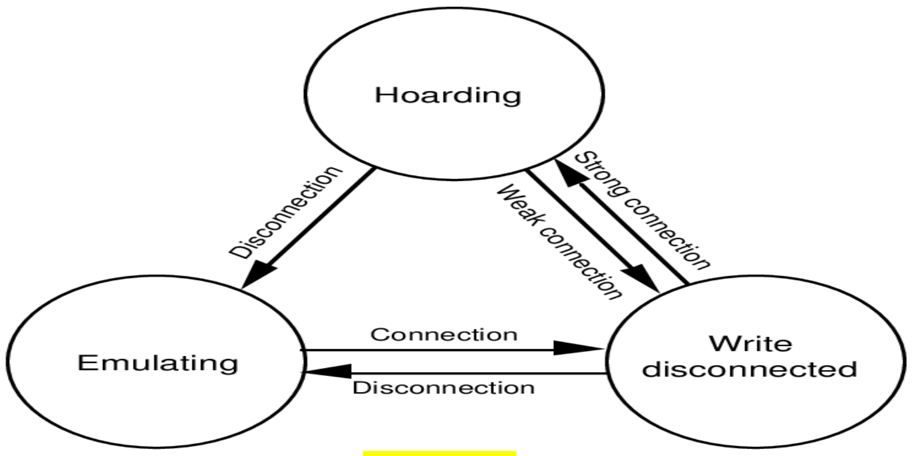
The function of size and bandwidth can hoard data whenever possible (if a network connection is available)-Hoarding
The user can pre-determine the priority file list
The content of the cache is determined by the list and LRU policy (least recently used)
Explicit prefetching possibilities
Regularly updated
Serving hoard/caching requests when disconnected-emulation
When the connection is regained, coordinate the repository/cache with the server and propagate the update – reintegrate/disconnect writing

File comparison:

Asynchronous, background
The system weighs update speed and minimizes network traffic

Cache miss:

User patient modelling: How long can the user wait for data without error messages?



Venus States
(Disconnected operation)

Hoarding:

Normal operation status (server is connected).
Venus will get useful data from the server to the client cache in preparation for disconnection.
Which data should be cached based on a priority algorithm, which consists of the most recent reference history and the storage database (HDB).
HDB is an entry for a path name that is used to identify objects that are explicitly stored (not cached) by the user.
When the cache is in a balanced state, Coda determines the priority of files placed in the cache.
Does the balance satisfy all these conditions?

- No un-cached files have higher priority than any cached files.
- The cache is full or there are no un-cached files with non-zero priority.

Emulation:

The state when disconnected.
Venus executes the server's jobs in the cache. It will record enough information in the log for replay during reintegration.
Each log contains system call parameters and the status of the objects referenced by the corresponding volume call.
If the client restarts, Venus also saves its cache and related data structures in non-volatile storage called Recoverable Virtual Memory (RVM).
RVM stores metadata for replay logs, HDB, cache directories, and status blocks.

Reintegration:

Restore the state of the server connection.
Venus updates the changes made in the emulation state to the server volume once.
Venus moves the replay log to ASVG in the server to be executed.
The replay algorithm is divided into 4 stages:

Phase 1: Parse the log and lock the reference object.
Phase 2: Verify the operation in the log.
Phase 3: Data transfer
Phase 4: Commit and release the lock.

Finally, Venus releases the replay log and resets the cache priority.

If there is a conflict (write/write) between the cache and the copy in the server, it will be resolved automatically or corrected manually according to the changed content.