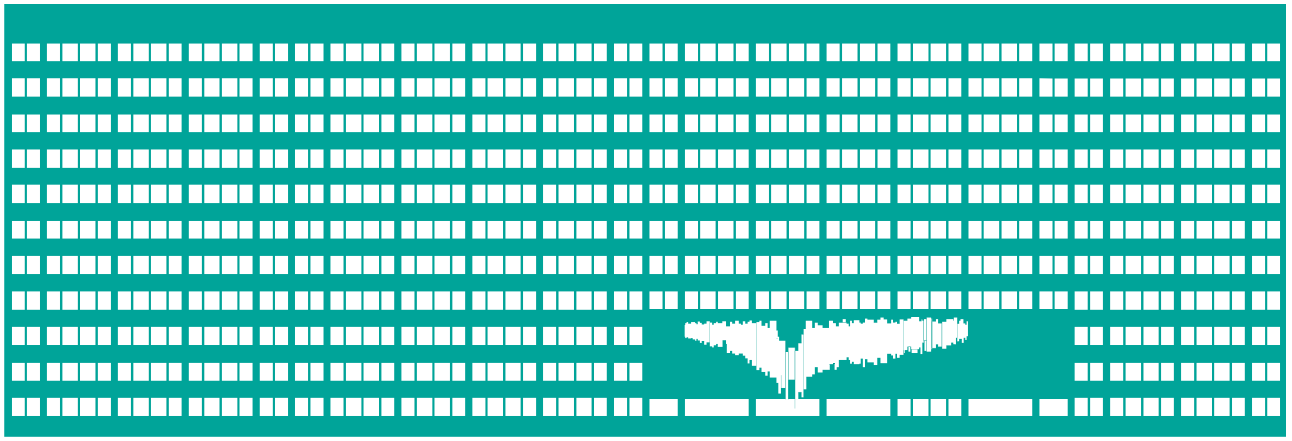
VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Mobile agents
System adaptation support



MS (Mobile Computing)
Lecture 3

Mobile agents

Mobile Agents (1)

- Composition of software and data which is able to migrate between nodes (e.g. objects – data, methods).
  - Processes along with data are dispatched from source node to accomplish a specified task. Proceeds autonomously and independently of the sending client after submission.
  - Upon reaching the server, it is delivered to an agent execution environment. AAA rules may apply. It can transport itself to another server, split new agents, or interact with other agents.
  - Upon completion, delivers the results to the sending client or to another server.

Mobile Agents (2)

- Features: autonomy, social ability, learning and mobility
- Mobile agents decide when and where to move
- A mobile agent accomplishes the move through data duplication. When a mobile agent decides to move it
    1. saves its own state
    2. transports saved state to the new host
    3. resumes execution from the saved state
- Support intermittent connectivity, slow networks, and light-weighted devices (however some common run-time environment must exist)

Mobile Agent advantages

- Computation bundles – relocatable data bundles, migrating client ↔ server, reducing network load.
- Dynamic adaptation – actions depend on the state of the host environment.
- Parallel processing – asynchronous execution on multiple heterogeneous network hosts.
- Network faults tolerant – ability to operate without an active connection between client and server
- Flexible maintenance - agent's actions must be updated only on the source host.

Mobile Agents properties

- Disconnected operation in mobile or fixed network
- Weak connectivity: the overall communication traffic is reduced to the submission of a single agent and its result
- Shift the brunt of computation from resource-poor mobile hosts to the fixed network when possible.
- It is possible to have multiple agents at different levels that cooperate (+inter-layer communication).
- Special mobile agents: *filters*, that operate on protocols (fewer exist than the number of applications) or OS level.

Examples of mobile agents

- Surrogate of the mobile host – functionality of the host is off-loaded to the agent
- Service-specific – implements the mobile-specific part of a application/service
  - file system agent
  - web agent
- Programmable – understands and can process code or other mobile agents sent by clients or servers
- Filter – implements the mobile-specific part of a protocol (e.g. changes the MPEG stream)

System support for mobile computing

System-level support

- Disconnections – autonomous disconnected operation of a mobile host
- Weak connectivity – tuned for expensive, low bandwidth environments, with high latency
- Mobility – basic support (e.g. new communication channels) & advanced support (e.g. migration of running processes and database transactions).
- *Failure recovery* – methods for (hard) failure handling and recovery.

Disconnected operation

Transition between following *states*:

- Data Hoarding – items needed for operation are preloaded into the mobile unit.
- Disconnected – only locally available data. Requests for other data may be queued & processed on reconnection.
- Reintegration – updates on a mobile host are reintegrated with other updates at fixed host (re-execution of the log).

Data hoarding state

- Hoarding approaches
  - Data relocation – data is inaccessible to other sites
  - Data replication or caching – consistency control
- Data objects type depends on the application and the underlying data model.
- Hoarding just before the disconnection, if we are expecting it. otherwise regularly (e.g. periodically)
- Future data need determination
  - users specify which data
  - past history of data accesses
  - based on application for which the system will be used

Disconnected state

- Applications with unsatisfied data requests can suspend execution or continue with some other job.
- Update approaches
    1. Pessimistic approach – updates only at one site (check-in/check-out $\times$ locking)
    2. Optimistic approach – updates at more sites (conflicts)
- Updates on mobile unit logged in client's stable storage.
- Keep the size of log small to save memory & reduce the time for update propagation and reintegration.
- Optimization during disconnected operation incrementally or as preprocessing step on reconnection.

Weak Connectivity

- Connectivity provided by slow or expensive networks. In wireless computing, connectivity varies in reliability, supplied bandwidth and cost.
- Intermittent connectivity – often lost for short periods.
- Systems, supporting adaptation to current degree of connectivity consider disconnected operation just the extreme case (total lack of connectivity).
- The aim of most proposals for weak connectivity is prudent use of bandwidth.
- Fidelity may be traded off against reduction in costs of communication.

Mobility

- Algorithms for dynamic data and task distribution
- Distinction between the replication cost in stationary and mobile computing – in mobile computing the I/O cost becomes insignificant, because of wireless com. charges.
- Dynamic data allocation algorithm: whenever a unit reads a copy, it is saved in a local DB. Whether to keep a (read-only) local copy of an item is chosen by one of following two different cost modules:
  - connection-based, where the user is charged per minute
  - message-based, where the user is charged per message

Mobility – data placement

- Alternative locations for placing copies (mainly to regard the search cost to locate copies at mobile hosts):
  - server
  - mobile client
  - location servers for the client
- Dynamic algorithms for replicated data placement – by letting transactions update the directory:
  - To locate copies, transaction first obtains read locks on the directory. A change in the placement of a copy is an update transaction on the directory.

Issues of computation move

- transparently from associated application
  × application-aware move
- location-dependent × location-independent
  (including movement of computation to and from a mobile client)
- after a computation unit has been completed
  × computation unit still in progress
- Inclusion of context transmission: enough information for the computation to continue correctly.
- How often does the computation move?
  - Involves various costs, including the cost of transferring the context and of informing other sites about the movement.

Mobility in various models

- Service hand-off: Surrogate × Service-specific.
- For mobile agents, primitives to create and move agents are provided by operating system or the programming language:
  - Tcl and Java move code
  - Telescript agents include execution state and are revived at each location
  - Obliq programming language maintains active network connections upon move

Failure recovery

- Emphasis on recording consistent global checkpoints
- Main issues to consider:
  - mobility of the mobile host from cell to cell: find the best place to store the next local checkpoint
  - is persistent storage available on the mobile host – determination of the degree of the mobile host participation in the checkpointing process
  - bandwidth, which in together with the availability of storage refines the mobile host participation in checkpointing process and affects the frequency of creating the checkpoints

Checkpoint protocols

- For correct recovery, the protocol must save a recoverable consistent global state.
- Consistency – a global checkpoint is *consistent* if for any message *m*, receive*(m)* is included in the global checkpoint, then send(*m*) is also included in it.
- Recoverability – to avoid loss of in-transit messages (messages that were sent but not received by any other process), if the global checkpoint contains the send(*m*) event but not the rcv(*m*) event then the checkpoint protocol must save message *m* as well.

Checkpoint protocol types (1)

- Coordinated protocols – participants have to coordinate their local checkpoints to ensure a consistent and recoverable global checkpoint
  - require sending control messages to different mobile hosts to synchronize the checkpointing process
  - search cost to locate the mobile host.
  - the mobile host might move to another cell before the checkpointing process is completed
  - during disconnection, the local checkpoint on mobile host is inaccessible to (synchronous) checkpointing algorithm

Checkpoint protocol types (2)

- Uncoordinated protocols – participants are permitted to checkpoint their local state independently.
  - During recovery, the effort to select one checkpoint from each participant must be coordinated to create a consistent global checkpoint.
  - Enables mobile participants to checkpoint their state without the need to exchange messages.
  - More information needs to be stored by checkpointing protocol to ensure execution in deterministic manner.
  - Additional messages have to be exchanged during recovery to find the global checkpoint, to gather information about other participants, or to garbage-collect stored information

Failure types

- Hard failures – handled by *hard checkpoints* stored in the fixed network
- Soft failures – handled by *soft checkpoints* stored locally on the mobile host.
  - Locally stored checkpoints don't consume bandwidth, are easy to create and allow mobile host to continue functioning during disconnections.
  - Soft failures do not permanently damage the mobile host, e.g. battery discharge or OS crashes

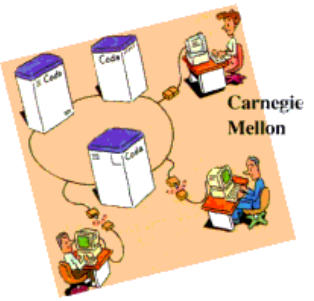
Checkpointing approaches

- Mobile host disk storage is considered unstable → inappropriate for storage of a participant's state.

- Mobile host has a relatively safe stable storage → it can participate in the checkpointing process like static hosts.
  - Current network characteristics are considered → a time coordinated recovery protocol which adapts its processing by tuning the MH's stable storage participation in the checkpointing process
  - Special treatment only for mobile host handoffs → participants are treated equally otherwise.

Distributed filesystems in mobile computing

CODA



(Illustration by Gaich Muramatsu)

CODA has been a prototype of a distributed filesystem with many features desirable for network filesystems, based on AFS filesystem:

1. Disconnected operation for mobile computing (*Venus*), network bandwidth adaptation, well defined semantics of sharing, even in the presence of network failures
2. Client side persistent caching, continued operation during partial network failures in server network
3. Server replication
4. Security model for authentication, encryption and access control
5. Weakly connected operations, isolation-only transactions

File system proxy

- Hides mobile issues from applications
- Emulates file server services on mobile computer
- CODA uses file system proxy to make existing applications work without any adaptation
- All updates are logged by the proxy during disconnection and upon reconnection, the log is replayed on file server.
- Automatic mechanisms for conflict resolution take place – optimistic concurrency control by proxy and file server.

Disconnected operation in FS

- Extended caching handling disconnections
  - cache misses cannot be served – treated as errors
  - updates at disconnected client won't be immediately propagated to the server
  - server can't notify the clients when updates occur
- Hoarding vs. prefetching
  - prefetching – ongoing process transferring soon-to-be-needed files to cache during periods of low network traffic with low overhead
  - hoarding – more critical than prefetching, client's need for data is overestimated (to a degree).

Hoarding in File systems

- Unit of Hoarding – differs based on the system
  - disk block
  - file
  - groups of files or directories
- Periodic process to ensure that critical files are in the cache – e.g. *hoard walk* in CODA
- A GUI can be used to assist the user to determine which files to hoard.

File selection for hoarding

- Explicit instructions from user
- Automatically from implicit information, usually based on the past history of file references.
- In CODA, priority prefetching based on a combination of recent file references (LRU) and user-defined hoard files.
- Tree-based method processes the history of file refs to build an execution tree – nodes are programs and files.
- Seer predictive caching – files are automatically pre-fetched based on the semantic distance (how closely related they are).
  - The measure is local reference distance between two files – the number of file references separating two adjacent references to them in the history.

Log Optimization in File Systems

- In CODA, a replay log records all relevant system call arguments & the version/state of all objects referenced by the call. Before a new record is appended:
  - Operation which overwrites the effect of earlier operations may cancel their corresponding records.
  - Inverse operation cancels both the inverting and inverted log records (e.g., *mkdir/rmdir*).
- Optimization at a preprocessing step is possible before reintegrating at reconnection. Two types of rules:
  - replacement rules remove adjacent redundant operations, e.g., *create* followed by a *move*.
  - ordering rules reorder adjacent operations to apply further replacement rules.

Update conflicts resolution

- Cache updates are considered tentative.
  - CODA – replay log is executed as a single transaction. All objects referenced in the log are locked.
- Different strategies for handling concurrent updates on files and on directories:
  - Directory resolution – fails only if a newly created name collides with an already existing name, if an object updated on the client or server has been deleted by the second side, or if directory attributes have been modified on the server and the client.
  - File resolution – based on application-specific resolvers (ASRs) per file.

Application-specific resolvers

- Programs encapsulating knowledge needed for conflict resolution, invoked at the client when divergent copies of the file are detected.
- Rules are used to select appropriate ASR. The ASR's mutations are executed locally on the client's cache and written back to the server atomically after finishing ASR.
- Execution of an ASR – guaranteed transaction semantics.
- If no ASR is found or the ASR execution fails, an error code indicating a conflict is returned. To resolve conflicts by user, a manual repair tool is run on the client.

Weak connectivity in FS – Cache (1)

- Cache Misses – services based on how critical the required item is vs. the currently available connectivity
- Cache Updates – frequency of propagation to the server must be considered.
  - Early reintegration cache updates approach:
    - reduces the effectiveness of log optimizations (records have less opportunity to be eliminated at the client)
    - affects the response times of other traffic
    - consistent cache management, timely update propagation and reduces the probability of conflicting operations.
    - keeps the small log in the client's memory → avoids the possibility of a client's cache overflow.
  - Update of Cached Objects – Notify the client each time an item is updated at the server × on demand, each time a client issues a read operation

Weak connectivity in FS – Cache (2)

- Cache Validation – intermittent connectivity → client must validate its cache.
  - Increase the granularity of cache coherency - each server maintains version stamps for volumes, i.e., sets of files, in addition to stamps for individual objects

  - Fetch-Only Operation – no continuous network connectivity, useful when the network charges per connection time (e.g., CSD, ISDN)
    - cache updates are deferred, no cache consistency protocol is used. The network is used only to satisfy cache misses.

Weak Connectivity implementation in CODA

- Cache misses are serviced selectively: a file is fetched only if the service time for the cache miss for the file (depending on bandwidth and other parameters) is acceptable by user

- Trickle reintegration – ongoing background process that propagates updates to servers asynchronously.
- Aging – record is not eligible for reintegration until it spends a minimal amount of time in the log (aging window)
- Reintegration chunk size is adaptive.

Other Weak Connectivity techniques in File Systems

- Three levels of queuing priority in the network drivers:
  - interactive traffic, other network traffic, & replay traffic
- Immediate propagation of file updates to the client through callbacks (locally updated directory in clients).
- Loose read – returns the value of the cache copy, if such a copy exists. Otherwise, returns the value of any copy × Strict read – returns the most recent value by contacting the necessary number of servers and clients.
- Pair-wise reconciliation – found mostly in peer-to-peer architectures:
  - Volume is compared to a single remote replica, runs till updates reach all sites storing replicas of the volume.