Mobile Computig: Case Studies

Evaggelia Pitoura

Computer Science Department,
University of Ioannina,
Ioannina, Greece
http://www.cs.uoi.gr/~ pitoura

Summer School, Jyvaskyla, August 1998

Case Studies

Four case studies:

- MIT's Rover distributed object development environment
- CMU's Coda file system
- Xerox's Bayou replicated storage system
- IBM's WebExpress web browsing system

Each one of these systems belongs to a different level of the software stack

System	Developed at:	Type	Architecture
Rover	MIT	Toolkit	Supports the development of client/server applications with mobile agents called RDOs
Bayou	Herox	Replicated Storage System	Peer-to-peer
Coda	CMU	File System	Replicated servers with a client-side proxy called Venus
WebExpress	IBM	Web Browsing System	Intercept Model

Case Studies: Rover

Rover[3] provides an application environment for the development of mobile applications.

A suite of software tools, including programming and communication abstractions.

Various applications have been developed on top of it including: a distributed calendar and an E-mail browser as mobile-aware applications, and a web browser as a mobile-transparent one.

System Architecture

Queued remote procedure calls (QRPCs): When a client issues an RPC, the RPC is stored in a local stable log and control is immediately returned to the application. The log is drained in the background, forwarding any queued RPCs to the server.

Relocatable dynamic objects (RDOs): an object encapsulating both code and data that can execute either at the client or the server. Application code and data are written as RDOs.

Applications split into two parts: one that will execute at the client and one at the server. These parts communicate through QRPCs.

The programmer links the application with the Rover toolkit. The application can then cooperate with the runtime system to import RDOs, invoke methods on the imported objects, and export logs of method invocations to the server.

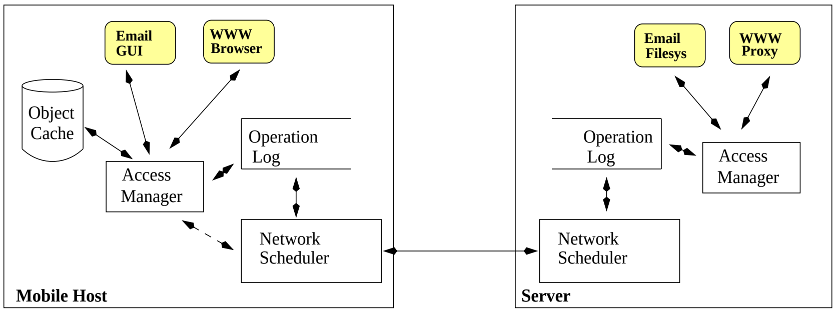
Case Studies: Rover

The *access manager* is responsible for all interactions among server-side and client-side applications. It services requests for objects (RDOs), mediates network access, log modifications to objects, and manages the client's object cache.

Client-side applications communicate with the access manager through QRPCs to import RDOs from severs and cached them locally in the *object cache*. Server-side applications are invoked by the access manager to handle requests from client-side applications.

Once an RDO has been imported into the client's cache, applications can invoke the methods provided by it. Method invocations without side effects are processed locally. At the application's discretion, method invocations with side effects also serviced locally, inserting tentative values in the object cache. In addition, operations with side effects insert a QRPC into a stable *operational log* at the client.

The *network scheduler* lazily transfers the log to the server-side and mediates between the various communication protocols and network interfaces.



Case Studies: Rover

Disconnected Operation

In anticipation of disconnections, useful RDOs are imported from the server to the client's object cache.

Prefetching is application specific. Rover applications provide prioritized prefetch lists based upon high-level user actions. For example, Rover Exmh, the TCL/TK-based E-mail browser ported in Rover's application space [3], automatically prefetches the user's inbox folder and any recently received messages, etc

Substantial but not exclusive support for *primary copy*, *tentative update* optimistic consistency. Each RDO has a "home" server that maintains the primary, canonical copy. Clients import secondary copies of RDOs into their local cache and export tentatively updated RDOs back to their home server.

During disconnection, Rover applications solely depend on the local cache. Requests for RDOs not in cache are queued as QRPCs in the operation log to be serviced upon reconnection.

To ease cache reconciliation at reconnection, in addition to new data values, Rover automatically logs method invocation as QRPCs in the operational log.

Thus, increase in the log size. Rover directly involves the application in the manipulation and reduction of the operation log. Applications can download procedures into their access manager to manipulate their log records.

Case Studies: Rover

During *reconnection*, updates are lazily propagated via QRPCs to the server.

When the QRPC for a mutating operation arrives at the server, Rover invokes this method on the primary copy. Update conflicts are detected and resolved by the server. The results of reconciliation always override tentative data stored at the object cache.

Weak Connectivity

By allowing the log to be incrementally flushed back to the server. Object updates in the operational log are lazily propagated to the server. Similarly, clients lazily fetch RDOs from servers in the local object cache, using QRPCs.

The network scheduler performs various optimizations on the operation log of QRPCs:

- reorders logged requests based on consistency requirements and application-defined operation priorities
- groups operations destined to the same server
- selects the appropriate transport protocol and medium over which to send them
- performs header and data compression.

Case Studies: Rover

By tightly coupling data and behavior into RDOs, application-specific and situation-specific optimizations.

For example, an RDO can include compression and decompression methods along with compressed data.

Through RDOs, Rover gives applications control over the partition of computation between the static and the mobile hosts: by importing RDOs from clients into servers and exporting updates.

For example, during weak connectivity, preferable to transmit to the client the code for implementing a particular GUI than transmitting the graphical display updates it generates.

Case Studies: Bayou

Bayou [13, 1, 14] is a replicated, weakly consistent storage system designed for a mobile computing environment that includes weakly connected portable machines.

Provides an infrastructure for building a variety of non-real-time collaborative applications such as shared calendars, mail and bibliographic databases, program development and document editing for disconnected workgroups.

System Architecture

Each data collection is replicated in full at a number of Bayou servers.

Clients can read-any and write-any copy residing on any server to which they can communicate.

A client and a server may be co-resident on a host, as would be typical of a laptop running in isolation.

Weak Connectivity

Read-any/write-any replication schema.

When a client submits a write to a server, the server performs the write only locally. Locally accepted writes are initially deemed tentative. The value of a tentative write is immediately made available for reading.

Case Studies: Bayou

Bayou servers propagate writes among themselves during pair-wise contacts, called *anti-entropy* sessions. During a session, the two servers exchange their writes so that at the end agree on the set of writes they have seen and on the order in which to perform them.

The Bayou system guarantees that: all servers will eventually receive all writes and that any two servers holding the same set of writes will have the same data contents.

To achieve that, writes are performed in the same global order at all servers and the procedures for detecting and resolving update conflicts are deterministic.

Need to undo thus a given write may be executed several times at a server. A write becomes *stable* when the server has received and executed all writes that precede it.

Bayou uses a *primary-commit* schema: one server designated as the primary takes responsibility for committing updates. Committed writes are ordered according to the times at which they commit at the primary and before any tentative writes. Knowledge of which writes have committed and in which order propagates to other servers during anti-entropy.

Each server maintains *two views* of the database: a copy that only reflects committed data and another full copy that also reflects the tentative writes currently known to the server.

Case Studies: Bayou

Dependency checks for automatic conflict detection and merge procedures for automatic conflict resolution with each write operation.

Dependency Check: an application-supplied query and its expected result. Before performing a write at a server, the corresponding query is run at the server against its current copy of the data. Conflict if the query does not return the expected result. In this case, the requested update is not performed and the server invokes a merge procedure.

Merge Procedure: program written in a high-level interpreted language responsible for resolving the conflict and producing a revised update to apply. When automatic resolution not possible, just logs the detected conflict to enable manual resolution.

Bayou allows replicas to remain accessible even when conflicts have been detected but not yet resolved.

A choice of reading committed or tentative data, and by setting an additional age parameter on reads.

Case Studies: Bayou

Session Guarantees

Clients can be attached to different servers.

A *session* is a sequence of read and write operations. Four *session guarantees* that can be applied independently:

Read Your Writes: read operations reflect previous writes.

Monotonic Reads: successive reads reflect a nondecreasing set of writes.

Writes Follow Reads: writes are propagated after reads on which they depend.

Monotonic Writes: writes are propagated after writes that logically precede them.

Either the storage system ensures them for each read and write operation belonging to a session, or else it informs the calling application that the guarantee cannot be met [13].

Disconnected Operation

By relying only on occasional pair-wise communication between servers, Bayou deals with arbitrary network connectivity.

Case Studies: Coda

Coda is a highly-available replicated file system.

Takes a transparent approach to mobility. Applications running on Coda use the standard Unix file system interface and can continue to run on mobile clients without any modification.

The client cache manager, *Venus* is solely responsible for coping with the consequences of mobility, acting as a client-side agent/proxy.

System Architecture

A number of clients and a much smaller number of file servers located at the fixed network [11, 4].

Volumes (i.e., groups of files) have replicas at more than one server. Replicas at servers are kept strictly coherent.

Clients cache data on their local disks. The replicas (i.e., cache copies) on clients are *second class*: less secure, incomplete, probably inaccurate and less persistent.

Disconnected Operation

Venus operates in one of three states: hoarding, emulating, and reintegration [4, 12].

Case Studies: Coda

During **normal operation**, Venus is in the hoarding state relying on servers but in alert for possible disconnection.

Cache coherence is based on *callbacks*.

Venus ensures that critical objects are cached using a priority-based cache management that combines:

–the recent reference history

– information in the form of a per-client hoarding database with entries are pathnames identifying objects of interest.

Venus periodically reevaluates which objects merit retention in the cache through *hoard walking*.

While **disconnected**, Venus relies solely on its cache.

Cache misses appear as failures to application programs and users.

An optimistic replica control strategy that allows updates in any partition.

Updates kept in an operation log, called CML, which is implemented on top of a transactional facility.

Optimizations [12, 10] to reduce the size of the CML: before a log is appended to the CML, Venus checks whether it cancels or overrides the effects of earlier records.

Case Studies: Coda

Upon **reintegration**, the cache is resynchronized with the servers.

Coda offers different strategies for handling concurrent updates on directories [5] and files [6].

For directories: resolution fails only if a newly created name collides with an existing name, if an object updated at the client or the server has been deleted by the other, or if the directory attributes have been modified at the server and at the client.

For files: application-specific resolvers (ASRs). An ASR is a program per file. It is invoked, when Venus detects divergence among its copies. The ASR is selected using rules specified by the user. If an ASR is found, it is executed on the client. The ASR's mutations are performed locally on the client's cache and written back to the server atomically after the ASR completes. While the ASR is being executed, the application that requested service for the file is blocked. Additional support is provided to ensure transactional semantics. If no ASR is found or the ASR's execution fails, a manual repair tool is then run on the client.

Isolation-only transactions (IOTs) [7] extend the Coda system so that it detects and resolves read/write conflicts. IOTs provide isolation but do not guarantee failure atomicity and only conditionally guarantee permanence.

Case Studies: Coda

Weak Connectivity

When weakly connected, Venus' behavior is a blend of its connected and disconnected mode behaviors [9].

- Updates are logged, as when disconnected; they are propagated to servers on the background.
- Cache misses are selectively serviced.
- Instead of callbacks, cache coherence is maintained by having the client validate its cache entries.

– *Trickle reintegration*: an ongoing background process to propagate updates at the client to the servers

Aging: a record is not eligible for reintegration until it has spent a minimal amount of time in the CML. This amount of time is called aging window (A) (results suggest a value of 10 min).

At the beginning of the trickle reintegration a logical divider, the *reintegration barrier*, is placed in the CML. During reintegration, the portion of the CML in front of the barrier is frozen. Only records at the back are examined for optimization.

Reintegrating all records older than A in one chunk could saturate a slow network. To bound the duration of degradation. the reintegration chunk size is made adaptive. If a file is very large, it is transfered in a series of fragments, each smaller than the currently acceptable chunk size.

Case Studies: Coda

– *Selective Service of Cache Misses*

User's *patience threshold* for a file: the maximum time that a user is willing to wait for that particular file The initial patience model is logarithmic to the hoard priority of the file. If for the current bandwidth, the estimated cache miss service time for a file is below its patience threshold, Venus services the miss transparently; otherwise Venus reports the miss by returning an error.

– *Cache Coherence*

To detect updates at the server, upon reconnection, the client must validate its cache.

Multiple levels for cache *granularity* [8]. A server maintains version stamps for each of its volumes, in addition to stamps on individual objects.

Mobility

A Coda client can transparently connect to any of the replicated servers

Case Studies: WebExpress

IBM's WebExpress [2] is a system for optimizing web browsing in a wireless environment

based especially on the predictability of running typical commercial transaction processing applications over wireless networks

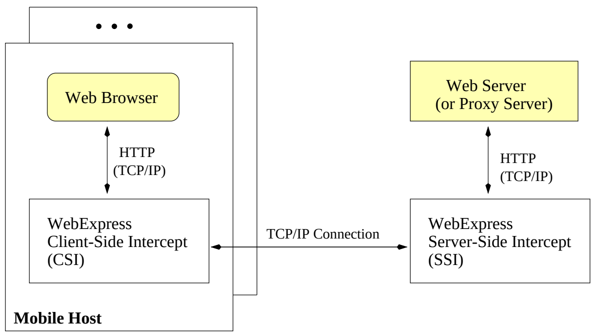
System Architecture

Mobile-transparent support to any web-applications

A pair-of-agent model called the *client/intercept/server* model: a client-side intercept agent (CSI) and a server-side intercept agent (SSI) inserted into the data path between the web client and the web server.

A web server can be a proxy server, a socket server, or the target web server. The CSI process runs on the end-user client mobile device, while the SSI process runs within the fixed network.

For scalability, the SSI supports multiple clients.



Case Studies: WebExpress

The CSI communicates with the web browser over a local TCP connection via the HTTP protocol.

Only change at the browser the specification the (local) IP address of the CSI as the browser's proxy address. The actual proxy or socket server address is specified as part of the SSI configuration.

The CSI communicates with an SSI process over a TCP connection using a reduced version of HTTP. The SSI reconstitutes the HTML data stream and forwards it to the designated web proxy server.

Likewise, for responses, or proxies, the CSI reconstitutes an HTML data stream received from the SSI and sends it to the web browser over the local TCP connection.

Connectivity and Weak Connectivity

– *Caching*

At both the CSI and the SSI.

The SSI cache is populated by responses from the requested web servers.

If the URL specifies an object in the CSI's cache, the object is returned, otherwise, the cache at the SSI is checked

Case Studies: WebExpress

The client cache replacement policy is an LRU (least recently used) policy augmented with an option that allows users to specify indefinite persistence of specific objects.

The server cache is also LRU managed and is designed to adapt to the browsing patterns of a set of users. Information retrieved by one user may be reused by others.

Objects loaded into the client's or server's cache persist across browser sessions. This increases the cache hit ratios but introduces cache coherency problems.

– *An age-based cache coherency method*

WebExpress associates a coherency interval (CI) with each cached object. The CI specifies when the object should be checked for changes and is set by each user or the administrator

When a cached object is referenced, the CSI checks whether the coherency interval has been exceeded. If not, the cached object is used. Otherwise, the CSI and SSI execute a protocol to determine whether a fresh copy of the object has to be fetched.

Case Studies: WebExpress

– *Differencing*

HTML byte streams that represent query responses to the same program often contain a lot of unchanging formatting data

A common base object is cached at both the CSI and SSI. When a response is received, the SSI computes the difference between the base object and the response and then transmits the difference to the CSI. The CSI merges the difference with its base object to create the browser response.

– *Single Connection*

In the normal HTTP protocol, the browser establishes a new TCP/IP connection for each image referenced in a document and for each selection of a hyper-link on the displayed page.

A single TCP/IP connection between the CSI and the SSI.

For each request received from the CSI, the SSI establishes a connection with the destination server and forwards the request. When the SSI receives the response from the server, it closes the connection with the server and sends the document to the CSI via the open TCP/IP connection without closing it. The CSI then forwards the document to the browser and closes its TCP/IP connection with the browser.

Case Studies: WebExpress

– *Header Reduction*

HTTP requests and responses are prefixed with headers that contain a list of MIME content-types

The CSI allows this information to flow in the first request after the CSI-to-SSI connection has been established. Both the CSI and the SSI save this list as part of the connection state information.

For each request received from the browser, the CSI compares the list received with its saved version.

Unlike the request headers, HTTP response headers are normally dissimilar. However, in general, only a few bytes (e.g., date time) vary among responses. Encoding the constant data (e.g., content-type) can reduce the response to just a few bytes.

Disconnected Operation

While disconnected, relies solely on the CSI cache.

To address lost connections, “asynchronous-disconnected” mode that permits requests to be automatically queued when connectivity is lost and resumed when connectivity is re-established.

multiple web requests without having to wait for their respective replies. Subsequently arriving responses are queued for the user to view

References

[1] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and B. Welch. The Bayou Architecture: Support for Data Sharing Among Mobile Users. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–7, Santa Cruz, CA, December 1994.

[2] B C. Housel, G. Samaras, and D. B. Lindquist. WebExpress: A Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment. *ACM/Baltzer Mobile Networking and Applications (MONET)*, 1997. Special Issue on Mobile Networking on the Internet. To appear. Also, University of Cyprus, CS-TR 96-18, December 1996.

[3] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers*, February 1997.

[4] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):213–225, February 1992.

[5] P. Kumar and M. Satyanarayanan. Log-based Directory Resolution in the Coda FIle System. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*, San Diego, CA, January 1993.

[6] P. Kumar and M. Satyanarayanan. Flexible and Safe Resolution of File Conflicts. In *Proceedings of the USENIX Winter 1995 Conference*, New Orleans, LA, January 1995.

[7] Q. Lu and M. Satyanarayanan. Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, Orcas Island, Washington, May 1995.

[8] L. Mummert and M. Satyanarayanan. Large Granularity Cache Coherence for Intermittent Connectivity. In *Proceedings of the 1994 Summer USENIX Conference*, Boston, MA, June 1994.

[9] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[10] B. Noble and M. Satyanarayanan. An Empirical Study of a Highly-Available File System. In *Proceedings of the 1994 Sigmetrics Conference*, Nashville, TN, May 1994.

[11] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[12] M. Satyanarayanan, J. J. Kistler, L. B. Mummert, M. R. Ebling, P. Kumar, and Q. Lu. Experience with Disconnected Operation in a Mobile Computing Environment. In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*, Cambridge, MA, August 1993.

[13] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session Guarantees for Weakly Consistent Replicated Data. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, pages 140–149, September 1994.

[14] D. B. Terry, M. M Theimer, K. Petersen, A. J. Demers, M. J Spreitzer, and C. H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.