Client-Server Computing in Mobile Environments

Jin Jing, GTE Laboratories Incorporated, 40 Sylvan Road, Waltham, MA 02454
and
Abdelsalam (Sumi) Helal, CISE Department, University of Florida, Gainesville, FL 32611
and
Ahmed Elmagarmid, Computer Sciences, Purdue University, W. Lafayette, IN 47907

Recent advances in wireless data networking and portable information appliances have engendered a new paradigm of computing, called *mobile computing*, in which users carrying portable devices have access to data and information services regardless of their physical location or movement behavior. In the meantime, research addressing information access in mobile environments has proliferated. In this survey, we provide a concrete framework and categorization of the various ways of supporting mobile client-server computing for information access. We examine characteristics of mobility that distinguish mobile client-server computing from its traditional counterpart. We provide a comprehensive analysis of new paradigms and enabler concepts for mobile client-server computing, including mobile-aware adaptation, extended client-server model, and mobile data access. A comparative and detailed review of major research prototypes for mobile information access is also presented.

Categories and Subject Descriptors: C.2.4 [**Computer–Communication Networks**]: Distributed Systems—*Wireless and Mobile Client/Server*

General Terms: Survey, Mobile Client/Server, Mobile Computing, Mobility Awareness, System Adaptation, Application Adaptation, Disconnected Operation, Data Dissemination, Caching, Cache Invalidation, Mobile Applications

Additional Key Words and Phrases: Survey, Mobile Computing, Client/Server, Mobile Data

---

1. INTRODUCTION

Advances in wireless networking technology and portable information appliances have engendered a new paradigm of computing, called *mobile computing*, in which users who carry portable devices have access to information services through a shared infrastructure regardless of their physical location or movement behavior. Such a new environment introduces new technical challenges in the area of in-

This is a preliminary release of an article accepted by ACM Transactions on Database Systems. The definitive version is currently in production at ACM and, when released, will supersede this version.

formation access. Traditional techniques for information access are based on the assumptions that the location of hosts in distributed systems does not change and the connection among hosts also does not change during the computation. In a mobile environment, however, these assumptions are rarely valid or appropriate.

Mobile computing is distinguished from classical, fixed-connection computing due to (1) the mobility of nomadic users and their computers and (2) the mobile resource constraints such as limited wireless bandwidth and limited battery life. The mobility of nomadic users implies that the users might connect from different access points through wireless links and might want to stay connected while on the move, despite possible intermittent disconnection. Wireless links are relatively unreliable and currently are two to three orders of magnitude slower than wire–line networks. Moreover, mobile hosts powered by batteries suffer from limited battery life constraints. These limitations and constraints leave much work to be done before mobile computing is fully enabled. This remains true despite the recent advances in wireless data communication networks and hand-held device technologies.

There has been a recent proliferation of research addressing issues of mobile systems and applications, especially for the purpose of mobile information access. In this survey, we attempt to provide a concrete framework and categorization of various methods of supporting such applications and information access from the viewpoint of client-server computing. The scope of this survey covers techniques and methods in support of components above the transport layer of networks. In a mobile client-server information system, a loose or tight collection of trusted information servers are connected via a fixed network to provide information services to a much larger collection of untrusted mobile clients over wireless and mobile networks.

1.1 Paradigms of Mobile Client-Server Computing

In this section, we briefly examine the impacts of mobility on information services and applications, and the new paradigms of client-server computing needed to deal with these impacts. A categorization of these computing paradigms is given below. This examination should facilitate our analysis and review of the various proposed techniques for mobile information access.

Existing research on mobile client-server computing can be categorized into the following three paradigms: (1) mobile-aware adaptation, (2) extended client-server model, and (3) mobile data access.

Mobile-aware adaptation: The dynamics of mobile environments and the limitations of mobile computing resources make adaptation a necessary technique when building mobile systems and applications. The paradigm of mobile-aware adaptation covers various strategies and techniques in how systems and applications respond to the environmental changes and the resource requirements. It also suggests the necessary system services that could be utilized by mobile-aware applications.

Extended client-server model: The extended client-server model facilitates mobile client-server information access. One distinguishing feature is the dynamic partitioning of client-server functionality and responsibilities. The extended client-server model provides a way to support the adaptation of mobile systems and applications. The paradigm of the extended client-server model includes various client-server computing architectures that enable the functional partitioning of applications between clients and servers.

Mobile data access: Mobile data access addresses issues such as how server data can be delivered to client hosts, how data over wireless and mobile networks is structured, and how the consistency of client cache is ensured effectively. The adaptive strategies for mobile data access depend largely on the type of communication links, the connectivity of mobile hosts, and the consistency requirements of applications. In our view, mobile data access provides another way to characterize the impact of mobile computing constraints on information access.

It should be noted that these new paradigms are closely related to each other. For example, the implementation for data delivery strategies and extended client-server architectures may involve the use of adaptation solutions. Extended client-server architectures might be needed to take advantage of new data delivery strategies. The categorization of new paradigms in this survey paper provides a comprehensive way to understand and analyze various proposed techniques in building mobile client-server information systems.

1.2 Organization of this Paper

The remainder of this paper is organized as follows. In Section 2, we discuss the paradigm of mobile client-server computing, namely, mobile-aware adaptation. The emphasis in this section is on understanding how adaptation strategies can be built into system and application components. In Section 3, we describe how the client-server model has been extended to adapt to the dynamic environments. In Section 4, we examine proposed techniques for mobile data access, including server data dissemination and client cache management. Finally, Section 5 reviews and analyzes three research prototypes of mobile client-server information systems, namely, Bayou, Odyssey, and Rover. Concluding remarks are offered in Section 6.

2. MOBILE-AWARE ADAPTATION

Mobile clients could face wide variations and rapid changes in network conditions and local resource availability when accessing remote data. In order to enable applications and systems to continue to operate in such dynamic environments, the mobile client-server system must react by dynamically adjusting the functionality of computation between the mobile and stationary hosts. In other words, the computation of clients and servers has to be *adaptive* in response to the changes in mobile environments [Katz 1994].

In [Satyanarayanan 1996], the range of strategies for application and system adaptation is identified, as shown in Figure 1. The range is delimited by two extremes. At one extreme, adaptation is entirely the responsibility of individual applications. This approach, called *laisse-faire adaptation*, avoids the need for system support. The other extreme, called *application-transparent adaptation*, places the entire responsibility for adaptation on the system. A typical case of this approach is to use proxies to perform adaptation on behalf of applications. Between these two extremes lies a spectrum of possibilities that are referred to as *application-aware adaptation*. This approach supports collaborative adaptation between the applications and the system. That is, the applications can decide how to best adapt to the

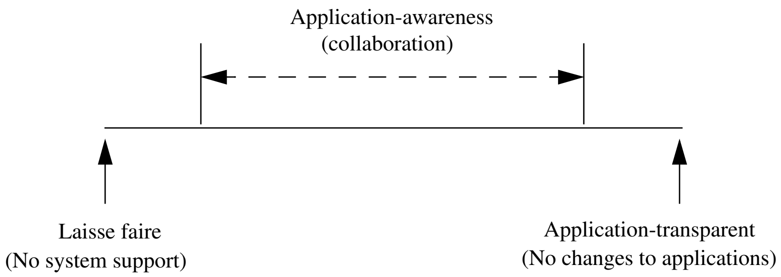

Fig. 1. Range of adaptation strategies

changing environment while the system provides support through the monitoring of resources and the enforcing of resource allocation decisions. This section will discuss different proposed adaptation approaches.

2.1 Application-Transparent Adaptation

Many existing client-server applications are built around the assumption that the environment of a client does not change. These applications are usually unaware of the mobility and make certain assumptions about the resource availability. The approach of application-transparent adaptation attempts to make these applications work with no modification in mobile environments. This is done by having the system shield or hide the differences between the stationary and mobile environments from applications. Examples of this approach include Coda [Satyanarayanan et al. 1990; Kistler and Satyanarayanan 1992], Little Work [Honeyman et al. 1992], and WebExpress [Housel and Lindquist 1996; Chang et al. 1997]. In these examples, a local proxy runs on the mobile host and provides an interface for regular server services to the applications. The proxy attempts to mitigate any adverse effects of mobile environments.

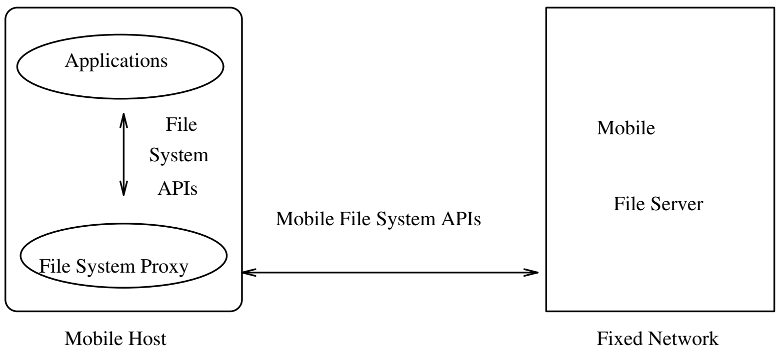

Fig. 2. File system proxy

2.1.1 *File System Proxy.* The basic idea is to use a file system proxy to hide mobile issues from applications and to emulate file server services on the mobile computers (see Figure 2). The Coda file system [Kistler and Satyanarayanan 1992], pioneering this approach, uses a file system proxy to make existing applications work with no modification. The proxy logs all updates to the file system during disconnection and replays the log on reconnection. Automatic mechanisms for conflict resolution using optimistic concurrency control are provided for directories and files through the proxy and the file server. The file system proxy in Coda facilitates the following features:

Disconnected Operations: A small collection of trusted Coda servers exports a location-transparent UNIX file name space to a larger collection of untrusted clients. On each client, a user-level process, *Venus*, manages a file cache on the local disk. Venus acts as a file system proxy and bears the brunt of disconnected operations. Venus operates in one of three states: hoarding, emulating, and reintegrating. In the hoarding state, server files are pre-fetched onto the mobile computer. Upon disconnection, Venus enters the emulating state and begins logging updates in a client modify log. In this state, Venus performs log optimizations to improve performance and reduce resource usage. Upon reconnection, Venus enters the reintegrating state, where it synchronizes its cache with the servers, propagates updates from the client modify log, and returns to the hoarding state.

In anticipation of disconnection, users may hoard data in the cache by providing a prioritized list of files in a per client hoard database. Venus combines the hoard database information with LRU (Least Recently Updated) information to implement a cache management policy. Periodically, Venus walks the cache to ensure that the highest priority items are present and consistent with the servers. A user may also explicitly request a hoard walk at any time. Since consistency is based on optimistic replica control, update conflicts may occur upon reintegration. The system ensures the detection and confinement of update conflicts and provides mechanisms to help the users recover from them.

Weakly Connected Operations: The file system proxy pre-fetches server data into the client cache and uses object or volume[1] callbacks for the cache validation in order to support weak connectivity. Volume call back is pessimistic in that invalidating a volume invalidates all the objects in this volume. However, the gain is in reducing cache invalidation information that needs to be communicated between the client and the server. The file system proxy can determine, based on factors such as cached data structures and connectivity changes, whether object or volume callbacks are best for a particular connection. The variable granularity of callback attempts to minimize the cost of validation and invalidation to provide effective support of operations for weakly connected clients.

Isolation-only Transactions: Disconnected operations may result in data inconsistency due to conflicting operations on multiple disconnected computers. Isolation-only Transaction (IOT) is proposed to automatically detect read/write conflicts. The execution of IOT is realized by the file system proxy code in the Coda system. An IOT provides consistency guarantees depending on the system connectivity conditions. Unlike traditional transactions, it does not guarantee failure atomicity and

[1] A volume is a collection of related objects.

only conditionally guarantees permanence.

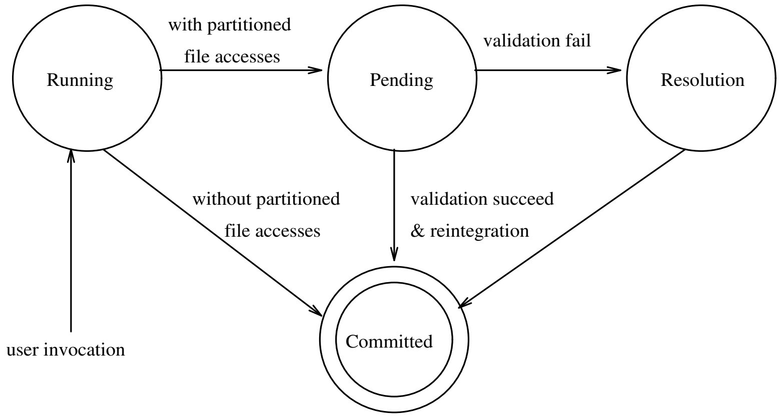

Fig. 3. A state transition diagram for IOT execution

When an IOT is completed, it enters either the committed or the pending state, depending on the connectivity condition (see Figure 3). If the execution of an IOT does not contain any partitioned file access, it is committed and its result is made visible on the servers. Otherwise, it enters the pending state for later validation. The result is temporarily held within the client's local cache and is visible only to subsequent processes on the same client. When the relevant partitions are repaired, the IOT is validated according to the isolation consistency criteria, namely, serializability. If the validation succeeds, the result will be immediately reintegrated and committed to the servers. Otherwise, the IOT enters the resolution state. When it is automatically or manually resolved, it will commit the new result to the server.

In addition to the Coda project, other projects address similar adaptation issues for mobile file system applications. In the Rover project, a file system proxy is added to the Rover Toolkit's object-based model [Joseph, Tauber, and Kaashoek 1996; Joseph, Tauber, and Kaashoek 1997]. It allows the Rover Toolkit to support both a file model and an object model for mobile applications. The file system proxy in Rover Toolkits also addresses a number of issues related to file caching, prefetching, and conflict detection and resolution. These issues are similar to those addressed by the Coda file system, except that they are associated with integrating a file system model with an object-based model. The Rover file system proxy consists of two components: a user-level installable local file system proxy located on the client and a remote file system proxy running on a Rover server. The two components work together with the use of Rover's queued communication mechanism that supports automatic message compression and batching. The Ficus file system is another file system supporting disconnected operations with application-transparent adaptation, but relies on version vectors to detect conflicts [Heidemann et al. 1992]. The Little Work project caches files for smooth disconnection from an AFS file system [Honeyman et al. 1992]. Conflicts are detected and reported to the user for manual resolution.

2.1.2 *Web Proxy.* Web proxy enables Web browsing applications to function over wireless links without imposing changes on browsers and servers. Web proxy can be used to prefetch and cache Web pages to the mobile client's machine, to compress and transform image pages for transmission over low-bandwidth links, and to support disconnected and asynchronous browsing operations.

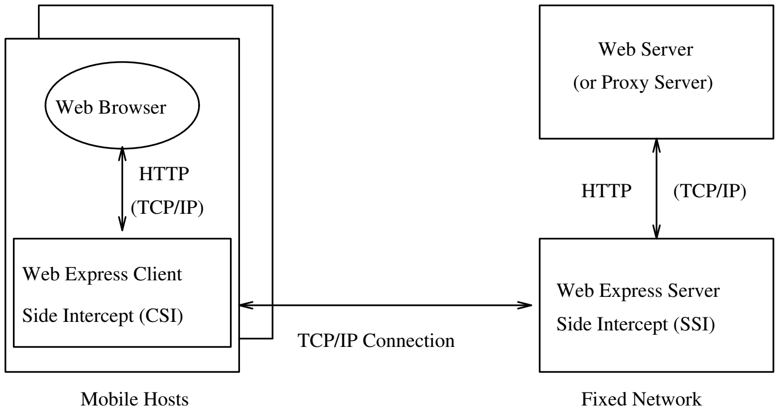

Fig. 4. The WebExpress intercept model

WebExpress [Housel and Lindquist 1996] uses this approach to intercept and control communications over the wireless link for the purposes of reducing traffic volume and optimizing the communication protocol to reduce latency. Two components are inserted into the data path between the Web client and the Web server: the Client Side Intercept (CSI) process that runs in the client mobile device and the Server Side Intercept (SSI) process that runs within the wired and fixed network (see Figure 4).

The CSI intercepts HTTP requests and, together with the SSI, performs optimizations to reduce bandwidth consumption and transmission latency over the wireless link. From the viewpoint of the browser, the CSI appears as a local Web proxy that is co-resident with the Web browser. On the mobile host, the CSI communicates with the Web browser over a local TCP connection (using the TCP/IP "loopback" feature) via the HTTP protocol. Therefore, no external communication occurs over the TCP/IP connection between the browser and the CSI. No changes to the browser are required other than specifying the (local) IP address of the CSI as the browser's proxy address. The CSI communicates with an SSI process over a TCP connection using a reduced version of the HTTP protocol. The SSI reconstitutes the HTML data stream and forwards it to the designated CSI Web server (or proxy server). Likewise, for responses returned by a Web server (or a proxy server), the CSI reconstitutes an HTML data stream received from the SSI and sends it to the Web browser over the local TCP connection as though it came directly from the Web server.

The proxy approach implemented in WebExpress offers the transparency advantage to both Web browsers and Web servers (or proxy servers) and, therefore, can be employed with any Web browser. The CSI/SSI protocols facilitate highly effective data reduction and protocol optimization without limiting any of the Web browser functionality or interoperability. WebExpress optimization methods are summarized below:

Caching: Both the CSI and SSI cache graphics and HTML objects. If the URL specifies an object that has been stored in the CSI's cache, it is returned immediately as the response. The caching functions guarantee cache integrity within a client-specified time interval. The SSI cache is populated by responses from the requested Web servers. If a requested URL received from a CSI is cached in the SSI, it is returned as the response to the request.

Differencing: CSI requests might result in responses that normally vary for multiple requests to the same URL (e.g., a stock quote server). The concept of differencing is to cache a common base object on both the CSI and SSI. When a response is received, the SSI computes the difference between the base object and the response and then sends the difference to the CSI. The CSI then merges the difference with its base form to create the browser response. This same technique is used to determine the difference between HTML documents.

Protocol reduction: Each CSI connects to its SSI with a single TCP/IP connection. All requests are routed over this connection to avoid the costly connection establishment overhead. Requests and responses are multiplexed over the connection.

Header reduction: The HTTP protocol is stateless, requiring that each request contain the browser's capabilities. For a given browser, this information is the same for all requests. When the CSI establishes a connection with its SSI, it sends its capabilities only on the first request. This information is maintained by the SSI for the duration of the connection. The SSI includes the capabilities as part of the HTTP request that it forwards to the target server (in the wire line network).

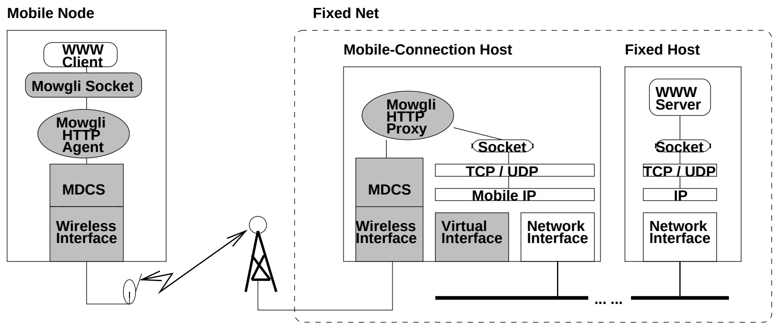

Fig. 5. The Mowgli wireless Web browsing architecture

The Mowgli project [Kojo, Raatikainen, and Alanko 1994] also uses a similar approach to support Web applications over wireless links (see Figure 5). A specialized HTTP agent and a specialized HTTP proxy are applied to: (1) reduce unnecessary message exchange, (2) reduce the volume of data transmitted over the wireless link, and (3) support disconnected operations. The HTTP agent acts as a local Web proxy on mobile client hosts while the HTTP proxy is located in the fixed network. With the agent-proxy approach, neither Web clients nor servers need to be modified. The HTTP agent intercepts requests generated by the Web client on the mobile node. It communicates with the HTTP proxy in the fixed network. Both the agent and the proxy maintain a cache of their own to improve performance over low-bandwidth links.

2.2 Application-Aware Adaptation

The approach of application-transparent adaptation does not require changing existing applications to run in mobile environments. However, it could sacrifice functionality and performance. As applications are shielded from dealing with mobility, it might be very hard for the system to make adaptation decisions that meet the needs of different and diverse applications. As a result, it may have to require some manual intervention by the user (e.g., having the user indicate which data to pre-fetch onto the mobile device) to make applications run smoothly. Such user-administered manual actions could be less agile to adapt to the changing environment. To address these problems, application-aware adaptation has been developed.

Application-aware adaptation allows applications or their extensions to react to the mobile resource changes. One way to realize the application-aware adaptation is through the collaboration between the system and individual applications. The system monitors resource levels, notifies applications of relevant changes, and enforces resource allocation decisions. Each application independently decides how best to adapt when notified. In a video player application, for example, such adaptation allows the video player system to scale back quality (and resource consumption) when application performance is poor and to attempt to discover additional resources by optimistically scaling up usage.

Depending on where the adaptive application logic resides, the approaches of application-aware adaptation can be divided into the following categories: *client-based application adaptation*, *client-server application adaptation*, and *proxy-based application adaptation*. The client-based adaptation allows the applications on mobile clients to react to the environmental changes, while client-server adaptation might have applications on both client and server to adapt to the changes. The proxy-based adaptation supports application-specific adaptation on the proxy server in the fixed networks. The application-specific proxies have been used as an intermediary between existing servers and heterogeneous mobile clients [Brooks et al. 1996; Brewer et al. 1998]. The proxies can perform aggressive computation and storage on behalf of mobile clients. These approaches can be complementary for a client-server information system. For example, both client-based and proxy-based adaptation can be used together in a single system to deal with the mobility.

2.2.1 *Client-Based Application Adaptation.* The Odyssey project [Noble et al. 1997] demonstrates a client-based collaborative adaptation approach for applications on mobile clients. In the collaborative adaptation, the system provides the mechanisms of adaptation, while the applications are free to specify adaptation policy. The division of responsibility addresses the issues of application diversity and concurrency. In a mobile information system, application data can be diverse in terms of data formats and consistency requirements. For example, the application data may be stored in one or more general-purpose repositories such as file servers, SQL servers, or Web servers. Alternatively, it may be stored in more specialized repositories such as video libraries, query-by-image-content databases, or back ends of geographical information systems. The application data can also have different dimensions for the specification and representation. For example, video data can have at least two specification dimensions: frame rate and image quality of individual frames. Spatial data, such as topographical maps, has dimensions of minimum feature size and resolution. Furthermore, concurrent applications can be very useful for mobile users. For example, an information filtering application may run in the background monitoring data such as stock prices and alert the user as appropriate. The collaborative adaptation accommodates the application diversity by allowing applications to determine how application data presented at a client matches the reference copy at the server based on resource levels. It also supports the application concurrency by allowing the system to retain control of resource monitoring and arbitration.

The application-aware adaptation in Odyssey is performed through the use of type-specific operations between the system and applications. The type-awareness is incorporated into both the system for efficient resource usage and the applications for differential handling of data types. The system-level knowledge of data types facilitates the optimization of the resource usage for different and diverse applications. For example, the size distribution and consistency requirements of data from an NFS server differ substantially from those of relational database records. Image data may be highly compressible using one algorithm, but not another. Video data can be efficiently shipped using a streaming protocol that drops rather than retransmits lost data; in contrast, only reliable transmissions are acceptable for file or database updates.

Odyssey incorporates type-awareness via specialized code components called *wardens*. A warden encapsulates the system-level support at a client to effectively manage a data type. To fully support a new data type, an appropriate warden has to be written and incorporated into Odyssey at each client. The wardens are subordinate to a type-independent component called the *viceroy*, which is responsible for centralized resource management. The collaborative relationship in the application-aware adaptation is thus realized in two parts. The first, between the viceroy and its wardens, is data-centric: it defines the consistency levels for each data type and factors them into resource management. The second, between applications and Odyssey, is action-centric: it provides applications with control over the selection of consistency levels supported by the wardens.

A number of similar approaches have also been discussed in the literature. In the Prayer system [Bharghavan and Gupta 1997], the application-aware adaptation is supported with the use of abstractions: QoS classes and adaptation blocks. A QoS class is defined by specifying the upper and lower bounds for resources. An application divides its execution into adaptation blocks. An adaptation block consists of a set of alternative sequences of execution, each associated with a QoS class. At the beginning of an adaptation block, an application specifies the QoS classes that it is prepared to handle, along with a segment of code associated with each class and an action to take should the QoS class be violated within the code segment.

In [Welling and Badrinath 1998], application-aware adaptation is implemented under an event delivery framework. In the framework, a notification subsystem, called event channel, delivers different events that are generated by the environment monitor to applications based on delivery policies. For example, a low memory event may be delivered to each application in series until enough memory is freed for other uses, while a low bandwidth event can be delivered to each application in parallel. The applications are notified of the events to react to the environmental changes.

2.2.2 *Client-Server Application Adaptation.* The Rover Toolkit [Joseph, Tauber, and Kaashoek 1996; Joseph, Tauber, and Kaashoek 1997] supports the application-aware adaptation through the use of relocatable dynamic object (RDO). The key task of the programmer when implementing an application-specific adaptation with Rover is to define RDOs for the data types manipulated by the application and for the data transported between the client and the server. The programmer divides the program that contains the RDOs into portions that run on the client and portions that run on the server; these parts communicate by means of queuing RPC. The programmer also defines the methods that update objects, including code for conflict detection and resolution.

At the level of RDO design, application designers have semantic knowledge that is very useful in implementing application-aware adaptation. By tightly coupling data with program code, applications can manage resource utilization more carefully than possible with a classic file or object system that handles only generic data. For example, an RDO can include compression and decompression methods along with compressed data in order to obtain application-specific and situation-specific compression, reducing both network and storage utilization. The RDO approach provides a generic framework to implement type-specific or application-specific operations for application-aware adaptation.

The application that consists of these RDO modules actively cooperates with the runtime system of the Rover Toolkit to import RDOs onto the local machine, invoke well-defined methods on those objects, export logs of method invocations on RDOs to servers, and reconcile the client's copies of the objects with the server's.

In [Davis et al. 1998], a tuple space based platform, called L2imbo, is proposed to support mobile distributed applications and adaptation. The platform offers an asynchronous programming model and architecture for reporting and propagating QoS information about mobile environments. The L2imbo architecture supports mechanisms of adaptation with the use of system and application agents that interact with the tuple space.

2.2.3 *Proxy-Based Application Adaptation.* The application-specific proxy has been proposed as an intermediary between clients and servers to perform computation intensive and storage intensive tasks, such as data type specific lossy compression, on behalf of clients [Brooks et al. 1996; Brewer et al. 1998; Zenel and Duchamp 1997]. This proxy architecture reduces the bandwidth demands on the infrastructure through lossy compression and allows legacy and other nonstandard (including thin) clients to inter-operate with existing servers. The proxy-based application adaptation allows the proxy agents to react to the environmental changes on behalf of mobile clients. This approach avoids inserting adaptation machinery at each origin server. From the client's perspective, the proxy is simply a server that gets the data from someplace else.

In the BARWAN project [Brewer et al. 1998], the application specific proxy uses the transcoders (i.e., proxy agents) to optimize the quality of service for the client in real time (see Figure 6). To use transcoding to adapt to network variation, the proxy must have an estimate of the current network conditions along the path from the proxy to the client. SPAND (Shared Passive Network Performance Discovery), a network measurement system, allows a measurement host to collect the actual application-to-application network performance (e.g., available bandwidth and latency) between proxies and clients. SPAND monitors end-to-end bandwidth and connectivity to the clients (and servers) and notifies the proxy of any changes, which may result in changes in transcoding to adjust the quality of service. The original servers are unaware of the transformations or of the limited capabilities of the clients or networks.

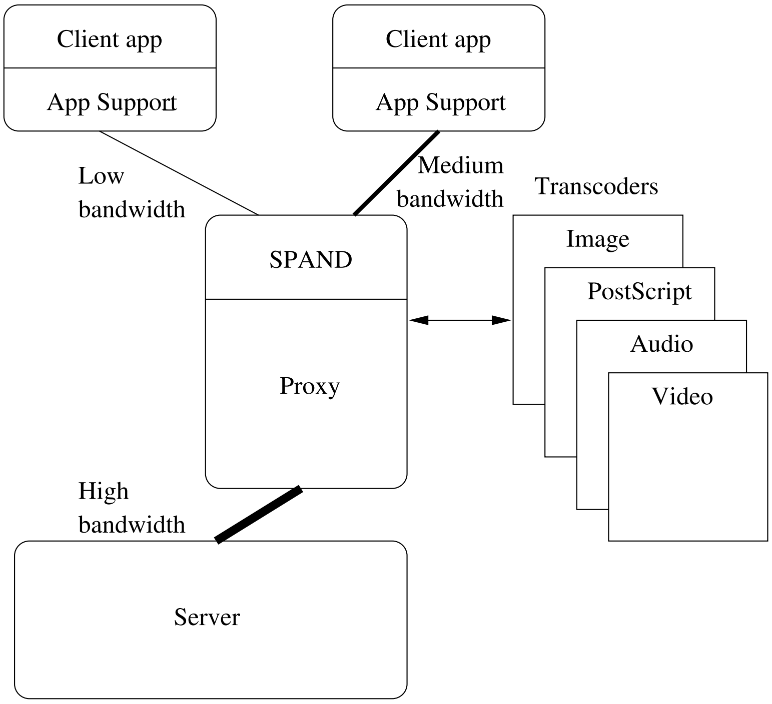

Fig. 6. A proxy-based adaptation architecture

Compact HTML [CompactHTML 1998] is a W3C submission of a standard for small information appliances. It defines a subset of HTML for small information appliances, such as smart phones, smart communicators, and mobile PDAs. The standard is intended as guidelines for manufacturers of small information devices, service providers, carriers, and software developers. Another similar language (which is relatively less compliant to the W3C HTML recommendation) is the Hand-held

Device Markup Language (HDML) [HDML 1997]. The HDML standard has been adapted into the Wireless Markup Language (WML) which makes up the application and presentation layers of the Wireless Application Protocol (WAP) stack, whose specification is being developed by the WAP forum [WAP 1996], as a *de facto* standard. WAP uses WML and provides third party proxies (via a session-layer protocol) as a means to negotiate device capabilities in terms of content characteristics. The WAP architecture is shown in Figure 7. The WAP proxies are capable of adapting to the mobile devices by negotiating and filtering the HTML Web content. Filtering is performed as a translation to and from a compact subset of the HTML language. The filtering process itself can be prescribed or automated, based on capability negotiation.

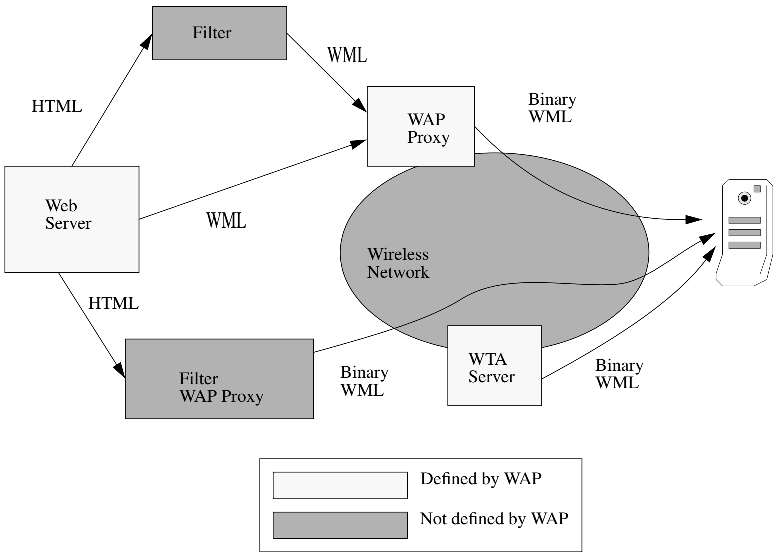

Fig. 7. The WAP architecture

3. EXTENDED CLIENT-SERVER MODEL

Another way to characterize the client-server computing in mobile environments is to examine the effect of mobility on the client-server computing model. In a client-server information system, a server is any machine that holds a complete copy of one or more databases. A client is able to access data residing on any server with which it can communicate. Classic client-server systems assume that the location of client and server hosts does not change and the connection among them also does not change. As a result, the functionality between client and server is statically partitioned. In a mobile environment, however, the distinction between clients and servers may have to be temporarily blurred [Satyanarayanan 1996], resulting in an *extended client-server model* shown in Figure 8. The resource limitations of clients may require certain operations normally performed on clients to be performed on resource-rich servers. Conversely, the need to cope with uncertain connectivity requires clients to sometimes emulate the functions of a server. An extreme case is called *the thin client architecture* that offloads most application logic and functionality from clients to stationary servers. In the thin client architecture, applications in stationary servers are usually mobile-aware and optimized for mobile client devices. The thin client architecture is especially suitable for dumb terminal or small PDA applications. The other extreme case is *the full client architecture*. The full client architecture emulates server functions on the client devices and, therefore, is able to minimize the uncertainty of connectivity and communications.

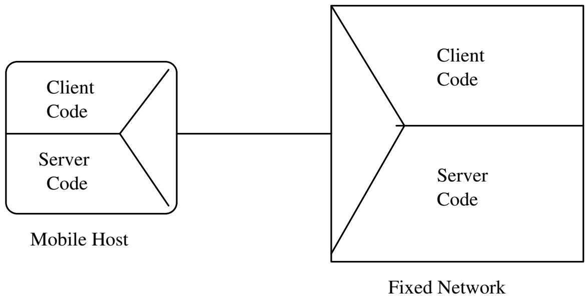

Fig. 8. Extended client-server model

3.1 Thin Client Architecture

The InfoPad project [Seshan et al. 1993] demonstrates the approach of thin client architecture. The InfoPad system is composed of four layers shown in Figure 9: Pad, InfoNet, Type Servers, and Applications. The Pad is a low power, portable multimedia terminal that is capable of displaying text and graphics, playing audio and compressed video, recording audio, and capturing pen input.

The InfoNet layer presents the software layer above the Pad with an abstraction in which each pad appears as a stationary, network-connected, multimedia terminal. This layer contains the routing algorithms to make mobility seamless and the routines to manage the wireless network resources (e.g., allocation of frequencies to pads).

The type servers make the pad appear as a typical workstation to applications; this allows for compatibility with standard workstation software. The type servers shield applications from knowledge of the mobile environments and terminal hardware. However, applications are optimized for use on the pad. The optimization takes advantage of the special characteristics of the pad, such as small screen size, lack of a keyboard, and support for handwriting and speech recognition.

While applications in the InfoPad system are customized for the characteristics of the pad, they do not contain code that allows them to dynamically adapt to changes in mobile networks. Instead, mobile-aware adaptation is largely performed in the InfoNet and type server layers. In this sense, the adaptation in the InfoPad system can be characterized as application-transparent adaptation rather than application-aware adaptation.

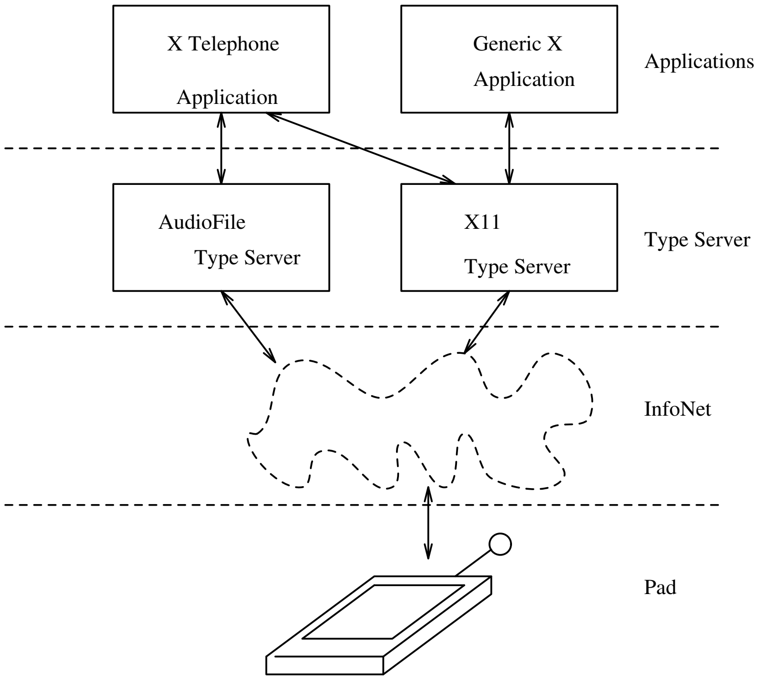

Fig. 9. InfoPad system layering

The thin client architecture from CITRIX Corporation allows a variety of remote computers, regardless of their platform, to connect to a Windows NT terminal server to remotely access a powerful desktop and its applications [CITRIX]. A server called MetaFrame runs under Windows NT in the desktop machine and communicates with the thin clients executing at the remote computers using the Independent Computing Architecture protocol (ICA). The ICA client and the MetaFrame server collaborate to display the virtual desktop on the remote computer screen. They also collaborate to process mouse and keyboard events, and to execute programs and view data stored at the server. All executions are remote and none takes place at the client portable computer. A research project at Motorola [Duran and Laubach 1999] extended CITRIX's thin client architecture so that it is optimized in the wireless environment. The work pointed out that bandwidth limitation is not as detrimental to the thin client performance as network latency. This is because the thin clients' use of bandwidth is limited.

W4 [Bartlett 1994] applies the technique of dividing application functionality between a small PDA and a powerful, stationary host for Web browsing. Other thin client examples include the Top Gun Wingman and Top Gun MediaBoard applications in the BARWAN project [Brewer et al. 1998].

3.2 Full Client Architecture

Mobile clients must be able to use networks with rather unpleasant characteristics: intermittence, low bandwidth, high latency, or high expense. The connectivity with one or more of these properties is referred to as *weak connectivity.* In the extreme case, mobile clients will be forced to work under the *disconnected* mode. The ability to operate in disconnected mode can be useful even when connectivity is available. For example, disconnected operations can extend battery life by avoiding wireless transmission and reception. It can reduce network charges, an important feature when charge rates are high. It allows radio silence to be maintained, a vital capability in military applications. Finally, it is a viable fallback position when network characteristics degrade beyond usability.

A full client architecture can be used to effectively support the disconnected or weakly connected clients. Compared to a thin client architecture, the full client architecture is at the other extreme of the range of extended client-server model. The full client architecture supports the emulation of functions of servers at the client host so that applications can be executed without fully connecting to remote servers. The emulation can be supported through a proxy or a "lightweight" server residing on client hosts. For example, a proxy can emulate some database operations to allow mobile users to work in a disconnected mode. Systems, enabling the full client architecture, include CODA and WebExpress.

3.3 Flexible Client-Server Architecture

Flexible client-server architecture generalizes both thin client and full client architectures in that the roles of clients and servers and application logic can be dynamically relocated and performed on mobile and stationary hosts (see Figure 10). In the flexible architecture, the distinction between clients and servers may be temporarily blurred for purposes of performance and availability. Furthermore, the connection between clients and servers can be dynamically established during the execution of applications (e.g., for the service handoffs).

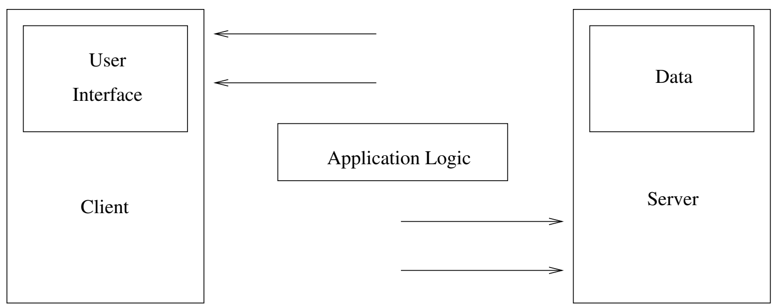

Fig. 10. A flexible client-server computing

3.3.1 *Mobile Objects.* Mobile objects (also known as mobile agents) are programming entities that can freely roam the network. Mobile objects enable client functions to run not only on mobile hosts, but also on stationary hosts as well. Furthermore, mobile objects allow clients to download the server code to the mobile host for execution. Mobile objects can contain threads and, therefore, be active. They can maintain state information and make intelligent decisions using it. They differ from downloadable applets in that they can independently roam among different machines and are not limited to being downloaded once from server to client. AgentTcl (developed at Dartmouth Colleage) [Gray 1995], Odyssey (developed by General Magic) [General Magic Odyssey 1997], and Aglets (developed by IBM) [IBM Aglets 1997] are examples of language tools that can be used to develop mobile objects.

One of the challenges to using mobile objects in wireless environments is how to implement the object transportation in a frequently disconnected or a weakly connected environment. The Rover Toolkit [Joseph, Tauber, and Kaashoek 1996; Joseph, Tauber, and Kaashoek 1997] provides mobile application support based on the mobile object idea. In the Rover Toolkit, a relocatable dynamic object (RDO) is an object (code and data) with a well-defined interface that can be dynamically loaded into a client computer from a server computer, or vice versa, to reduce client-server communication requirements.

3.3.2 *Collaborative Groups.* A commonly occurring case may be several users disconnected from the rest of the system while actively collaborating; a canonical example is a group of colleagues taking a business trip together. Rather than giving the members of this disconnected working group access only to the data that they had the foresight to copy to their personal machine, a collaborative group architecture grants any group member access to any data that is available to the group.

The architecture is based on a division of functionality between servers, which store data, and clients, which read and write data managed by servers. A server is any machine (e.g., a mobile host) that holds a complete copy of one or more databases. A client is able to access data residing on any server to which it can communicate; and conversely, any machine holding a copy of a database, including personal laptops, should be willing to service read and write requests from other nearby machines.

In this architecture, portable computers can be servers for some databases and clients for others. The Bayou system [Demers et al. 1994; Terry et al. 1995] is an example that implements the collaborative group architecture.

3.3.3 *Application-Specific Proxy.* The application-specific proxy acts as an intermediary between clients and servers to perform computation-intensive and storage-intensive tasks on behalf of clients. An application-specific proxy on a stationary host supports proxy agents (which can be fixed or mobile) to dynamically "transcode" or "distill" application data to reduce the bandwidth consumption between the proxy and the mobile client. This proxy allows legacy and other non-standard (including thin) clients to inter-operate with existing servers. From the client's perspective, the proxy is simply a server that gets the data from someplace else. The examples of application-specific proxies are discussed in [Brooks et al. 1996; Brewer et al. 1998; Zenel and Duchamp 1997].

3.3.4 *Virtual Mobility of Servers.* In a wireless information system, information (data) servers are connected via fixed networks to provide information services to mobile users. The replication (or partition) of information services could help reduce the latency of remote operations and balance the workloads of data servers in a distributed and multiple networks environment. The replication of information services in different networks can be used to support *the application service handoffs* for moving mobile clients.

In a wireless environment, the client moves around, perhaps to areas it has never visited before; once in a new milieu (or a new network environment), it will negotiate with some near-by machine to become a new coordinator to continuously provide services for its operations. The movement of mobile hosts may result in a long path of communications in fixed networks because the physical distance may not necessarily reflect network and service distance between the client and the server, especially when the movement crosses the boundary of different networks. The long path of communication in fixed networks will increase the traffic and latency of transactional services. If the new coordinator could always use a nearby or local information service site as the client's data server, the traffic and latency in fixed networks can be reduced for the continuous interaction between the client and the information service server. Therefore, the mobility of the client introduces the concepts of *virtual mobility of servers* and application service handoffs.

An issue of supporting the virtual mobility of servers is to minimize the overhead of application service handoffs for synchronous multi-server operations. The work in [Tait and Duchamp 1991; Tait and Duchamp 1992] investigates how to maintain replicas in a distributed file system that supports mobile clients. This work assumes that (1) clients' movements cannot be constrained, although patterns of movement may exist; (2) the latency of remote operations increases as the distance between hosts increases; (3) sequential sharing workload is not uncommon, but simultaneous sharing (other than read-read) workload is rare; and (4) a file cache of modest size is maintained by each client. The design goals in this work include: minimizing synchronous multi-server operations; easing the addition and deletion of server sites; and allowing for incomplete replication.

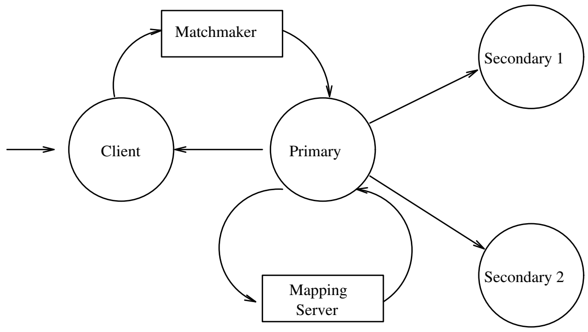

Fig. 11. The primary-secondary server hierarchy

To achieve these goals, a primary-secondary server hierarchy architecture is proposed, as shown in Figure 11. Typically, the client need not contact any (remote) server, but it communicates synchronously with the (local) primary server only. All updates are stored in the client's cache. The primary server makes periodic pickups from the clients it is currently servicing and propagates updates back to the secondary servers asynchronously. This pickup strategy allows the client's write operation to return immediately after placing the new value in its cache.

The primary servers are used as intermediaries between clients and secondary servers. Because the system supports replica addition/deletion and incomplete replication, a primary server must learn about mapping from the file system to the secondary servers. The mapping is provided by calling a *mapping server* function specified by the client. For example, someone from New York who is visiting Seattle would call the Seattle matchmaker to obtain a local primary server site. The locally-chosen primary server then calls the mapping server in New York to locate files that the client wishes to access. After the handoff of primary servers, the new primary server can lazily copy the client's file from the old cache in the old primary server. Therefore, this method always connects the mobile client to the local primary server for information access in a distributed file system.

The work in [Krishnakumar and Jain 1994; Jain and Krishnakumar 1994] presents a system architecture and application service handoff protocols for virtual mobility of servers. The architecture is based on replicated distributed servers connected to users via a personal communication services (PCS) network. Under this architecture, as the user moves out of one service area into another, the local server at the new service area takes over providing the service. This service handoff for the virtual mobility of the server is broadly analogous to the PCS call handoff procedure and also requires that service continuation is transparent with no interruptions.

For the application services handoffs, a service coordinator, when informed that the user has moved, initiates the set-up of a conference call between the current server, the mobile host, and the new server so that the service can be transparently handed off to the new server. The coordinator can then terminate the call with the old server.

Before the new server takes over during a service handoff, it has to know what the mobile user is currently doing with the service, i.e., the *context* of the user with respect to the service. Context information is the information associated with a user and a service so that the user can access different servers transparently. Part of the context is static, including password and access rights that do not change as the user accesses information. However, the context also includes dynamic session-specific data information such as how much data has been read or modified by the user, whether the changes are meant to be transactional, whether the user holds any locks to access the data, and so on. In [Elmagarmid, Jing, and Bukhres 1995], an approach called *Reservation Algorithm* (RA) is proposed to avoid transaction log transfer and the use of global commit protocol.

4. MOBILE DATA ACCESS

Mobile data access enables the delivery of server data and the maintenance of client-server data consistency in a mobile and wireless environment. Efficient and consistent data access in mobile environments is a challenge research area because of weak connectivity and resource constraints. The data access strategies in a mobile information system can be characterized by delivery modes, data organizations, and consistency requirements, etc. The mode for server data delivery can be server-push, client-pull, or hybrid. The *server-push delivery* is initiated by server functions that push data from the server to the clients. The *client-pull delivery* is initiated by client functions which send requests to a server and "pull" data from the server in order to provide data to locally running applications. The *hybrid delivery* uses both server-push and client-pull delivery. The data organizations include mobility-specific data organizations like mobile database fragments in the server storage and data multiplexing and indexing in the broadcast disk. The consistency requirements range from weak consistency to strong consistency. Figure 12 illustrates the new paradigm of mobile data access for mobile information access. In this section, we will examine various proposed approaches that offer the new paradigm of data access in mobile client-server information systems.

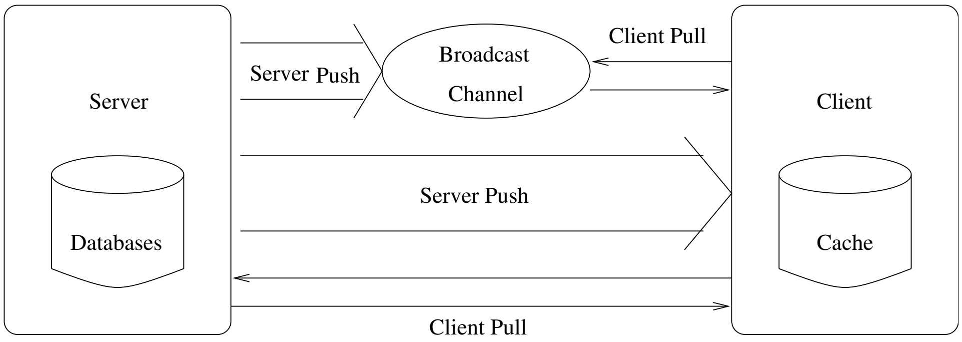

Fig. 12. A mobile data access paradigm

4.1 Server Data Dissemination

In many applications (e.g., Web access), the downstream data volume from servers to clients is much greater than the upstream data volume from clients back to servers. The unbalanced communications are referred to as asymmetrical communications between clients and servers. Application examples of asymmetrical communications in wireless environments include Hughes's DirectPC (www.hns.com) and CAI's Wireless Internet Access (www.caiwireless.net), where clients at mobile hosts usually have a lower bandwidth cellular or PSTN link while servers at fixed hosts may have relatively high bandwidth broadcast capability.

A challenge problem in supporting applications with asymmetrical communications is how to deliver server data and information to a large number of clients. To address this scalability problem, a new information system architecture that exploits broadcast-based dissemination capability of communications has been proposed [Herman et al. 1987; Imielinski and Badrinath 1994; Acharya et al. 1995; Acharya, Franklin, and Zdonik 1997; Su and Tassiulas 1998]. The central idea is that the servers exploit the downstream communication capacity in bandwidth by *broadcasting* data to multiple clients. This arrangement is called a push-based architecture where data is pushed from the server to the clients. In contrast, most traditional client-server information systems use pull-based data delivery to provide data to locally running applications.

4.1.1 *Broadcast Disks.* When a server continuously and repeatedly broadcasts data to a client community, the broadcast channel becomes a "disk" from which clients can retrieve data as it goes by. The broadcasting data can be organized as *multiple disks* of different sizes and speeds on the broadcast medium [Acharya et al. 1995]. The broadcast is created by multiplexing chunks of data from different disks onto the same broadcast channel. The chunks of each disk are evenly interspersed with each other. The chunks of the fast disks are repeated more often than the chunks of the slow disks (see Figure 13). The relative speeds of these disks can be adjusted as a parameter to the configuration of the broadcast. This use of the channel effectively puts the fast disks closer to the client while at the same time pushing the slower disks further away.

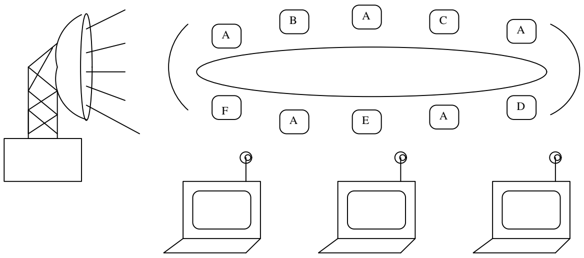

Fig. 13. A broadcast program

This technique presents an opportunity to more closely match the broadcast to the workload at the client. Assuming that the server has an indication of the client access patterns (either by watching their previous activity or from a description of intended future use from each client), then hot pages or pages that are more likely to be of interest to a larger part of the client community can be brought closer while cold pages can be pushed further away. This, in effect, creates an arbitrarily fine-grained memory hierarchy, as the expected delay in obtaining an item depends upon how often that item is broadcast. The broadcast disk technique, therefore, provides improved performance for non-uniformly accessed data.

In the simplest scenario, the server can broadcast different items at the same frequency. With the "flat" broadcast, the expected delay required prior to obtaining an item is the same for all items broadcast (namely, half a broadcast period) regardless of their relative importance to the clients. This "flat" approach has been adopted in earlier work on broadcast-based information system such as Datacycle [Herman et al. 1987] and the work in [Imielinski, Vishwanath, and Badrinath 1994a; Imielinski, Vishwanath, and Badrinath 1994b]. By comparison, the server can broadcast different items with differing frequency; important items can be broadcast more often than others.

Similar to the broadcast disk concept, a pyramid broadcasting method is used to provide Video-On-Demand services to mobile users [Vishwanath and Imielinski 1995]. In pyramid broadcasting, the most frequently requested movies are multiplexed on the broadcast network, resulting in radical improvement of access time and efficient bandwidth utilization.

4.1.2 *Indexing on Air.* In the "push-based" approach, servers periodically broadcast most frequently requested data items (hot spots). The server should dynamically adjust the content of the broadcast hot spot, depending on the periodically measured demand distribution. The client is lazy in that it transmits only when necessary. The client could also stay in the doze mode (turning the power off) as much as possible. To minimize wake-up time, clients can use selective tuning capabilities in the broadcasting channel. In [Imielinski, Vishwanath, and Badrinath 1994a; Imielinski, Vishwanath, and Badrinath 1994b], the authors discuss basic methods for data organization on the broadcast channel that provide selective tuning capabilities for clients. In these methods, index information is broadcasted together with the data. The index is structured so that the following two parameters are minimized:

—*Query Time*: Amount of time that it takes for a client to issue a query until the answer is received by the client.

—*Listening Time*: Amount of time spent by the client listening to the channel.

Query time is proportional to the overall size of the broadcast data. Therefore, the presence of the index increases the query time since the presence of the index increases the overall broadcast size. However, the presence of the index reduces the listening time. This, in turn, reduces energy consumption because clients can use the selective listening capabilities in broadcasting channels to stay in the doze mode and minimize wake-up time, as demonstrated in [Imielinski, Vishwanath, and Badrinath 1994a; Imielinski, Vishwanath, and Badrinath 1994b].

4.2 Client Cache Management

Caching of frequently-accessed data items is an important technique that reduces contention and improves query response times on narrow bandwidth wireless links. The cached data can also support disconnected or intermitted connected operations. However, cache pre-fetching and consistency strategies can be greatly affected by the disconnection or weak connectivity of mobile clients. The weak connectivity makes cache coherence expensive due to communication latency and intermittent failures. Pre-fetching (or *hoarding*) data into the client cache prior to disconnection is a difficult challenge in mobile client-server computing. This subsection describes an automated hoarding approach and two cache validation mechanisms.

4.2.1 *Automated Hoarding.* A useful solution to support disconnected operations is hoarding, in which non-local files are cached on the client cache prior to disconnection. The difficult issue for hoarding is which files should be selected and stored locally. Possible solutions include choosing the most recently referenced files or asking the user to participate at least peripherally in managing hoard contents. The former approach might be wasteful of scarce hoard space, while the latter requires more expertise and involvement that most users are willing to offer.

In the SEER system, an automated predictive hoarding approach is developed [Kuenning and Popek 1997]. The automated predictive hoarding is based on the idea that a system can observe user behavior, make inferences about the semantic relationships between files, and use those inferences to aid the user. In SEER, an observer component watches the user's behavior and file accesses, classifying each access according to type, converting path names to absolute format, and feeding the results to a correlator component. The correlator component evaluates the file references, calculating the semantic distances among various files. These semantic distances drive a clustering algorithm that assigns each file to one or more projects. When new hoard contents are to be chosen, the correlator component examines the projects to find those that are currently active, and selects the highest-priority projects until the maximum hoard size is reached. In this way, SEER is able to operate without user intervention (though the user might involve informing the computer that a disconnection is imminent).

The fundamental assumption of SEER is that there is semantic locality in user behavior. By detecting and exploiting this locality, a system can make inferences about the relationships between various files. Once these relationships are known, it is possible to automate the hoarding. To detect semantic locality, SEER uses a concept known as semantic distance. Conceptually, semantic distance attempts to quantify a user's intuition about the relationship between files. A low semantic distance suggests that the files are closely related and thus are probably involved in the same project, while a large value indicates relative independence and different projects.

The semantic distance is based on measurements of individual file references, rather than looking at the files themselves. The distance between references is then summarized to produce a value that is relevant to the individual files. Several semantic distance measurement methods are defined based on file references in [Kuenning and Popek 1997].

4.2.2 *Varied Granularity of Cache Coherence.* Consistency methods in traditional client-server architecture can be divided into two categories: (1) *callback approach* when servers send invalidation messages directly to the clients that have cached the data items to be updated and (2) *detection approach* when clients send queries to servers to validate cached data. The difficulty of using these traditional methods in mobile environments is due to the disconnection and weak connectivity of clients. Frequently disconnected clients make it very ineffective to use the detection approach. On the other hand, the classic callback approach may also be very expensive, due to network latency or intermittent failures. After a long disconnection, many data items at the server side may have been updated. In this case, the time for the callback invalidation of each data item can be substantial on a low network.

In the Coda system [Satyanarayanan et al. 1990; Kistler and Satyanarayanan 1992; Mummert and Satyanarayanan 1994], clients can track the server state at multiple levels of granularity. A server maintains version stamps for each of its file volumes, in addition to stamps on individual objects (or items). When an object is updated, the server increments the version stamp of the object and that of its containing volume. Clients cache volume version stamps in anticipation of disconnection.

When connectivity is restored after a network failure, the client presents volume stamps for validation. If a volume stamp is still valid, then so is every object cached from the volume. If a volume stamp is not valid, cached objects from the volume need to be validated individually. Even in this case, performance is no worse than in the original scheme [Mummert and Satyanarayanan 1994]. On the other hand, if the validation for each individual data item is not possible due to slow connections, the client can assume that all items in the volume are invalid or defer the validation until the time when the item is used. The experiments and measurements from the Coda system confirm that the varied granularity of cache coherence dramatically improve the speed of cache validation [Mummert and Satyanarayanan 1994].

4.2.3 *Cache Invalidation Reports.* A dissemination-based approach to the problem of invalidating caches in wireless environments is proposed in [Barbara and Imielinski 1994] by utilizing wireless broadcast channels. In this approach, a server periodically broadcasts an *invalidation report* that reports data items which have been changed. Rather than querying a server directly regarding the validation of cached copies, clients listen to these invalidation reports over broadcast channels. This approach is attractive because a server does not need to know the location and connection status of its clients, and the clients do not need to establish an "uplink" connection to the server to invalidate their caches.

Three algorithms are presented in [Barbara and Imielinski 1994], namely, the Broadcasting Timestamps (TS), Amnesic Terminals (AT), and Signatures (SIG). In the TS algorithm, the invalidation report includes only the information regarding the data items that have been updated within the preceding w seconds. The report includes the names of these items and the timestamps at which they were updated. The invalidation report in the AT algorithm includes the identifiers of data items that were updated during the last broadcast period L. In both the TS and AT algorithms, clients must invalidate their entire cache when their disconnection period exceeds a specified length (w seconds for TS and L seconds for AT). In the SIG algorithm, the report contains a set of combined signatures of data items.[2] The structure and size of these signatures are designed to diagnose up to f differing items. If more than f different items (it does not matter whether these items had been cached or not) are updated in the data server since the combined signatures were last cached, most items cached by the clients will be invalidated by the SIG algorithm, although many are in fact valid.

An improved invalidation report method called *Bits-Sequences* (BS) is proposed in [Jing, Elmagarmid, Helal, and Alonso 1997]; this method adapts to long disconnected clients. In the Bit-Sequences (BS) algorithm, the server broadcasts a set of bit sequences. Each sequence consists of a series of binary bits and is associated with a timestamp. Each bit represents a data item in a database. A bit "1" in a sequence means that the item represented by the bit has been updated since the time specified by the timestamp of the sequence. A bit "0" means that that item has not been updated since that time.

Clients must check the invalidation report before they can use their caches for query processing. If a bit sequence among the sequence set with the most recent timestamp equals to or predates the disconnection time of the client, the sequence

[2]Signatures are checksums computed over the value of data items in the database. A combined signature is the Exclusive OR of a set of individual signatures. Each combined signature, therefore, represents a subset of data items. In the SIG algorithm, m combined signatures are computed such that an item i is in the set of combined signature S_j $(1 \leq j \leq m)$ with probability $\frac{1}{f+1}$ where f is the number of differing items up to which the algorithm can diagnose.

will be used to invalidate its caches. The data items represented by these "1" bits in the sequence will be invalidated. If there is no such sequence (i.e., the disconnection time precedes the timestamp of the highest sequence), the entire cache in the client will be invalidated.

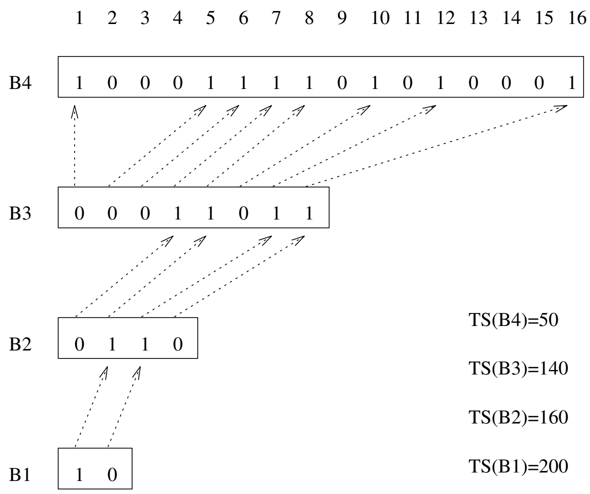

Fig. 14. A Bit-Sequences example

Consider a database consisting of 16 data items. Figure 14 shows a Bit-Sequences (BS) structure reported by a server at the time 250. Suppose a client listens to the report after having slept for 80 time units. That is, the client disconnected at the time 170 (=250-80), which is larger than $TS(B_2)$ but less than $TS(B_1)$. The client will then use B_2 to invalidate its caches. To locate the items denoted by the two "1" bits in B_2, the client will check both B_3 and B_4 sequences, using the following procedure. To locate the second bit that is set to "1" in B_2, check the position of the second "1" bit in B_3. The second "1" bit in B_3 is in the 5th position; therefore, check the position of the 5th "1" bit in B_4. Because B_4 is the highest sequence and the 5th "1" bit in B_4 is in the 8th position, the client concludes that the 8th data item was updated since the time 170. Similarly, the client can deduce that the 12th data item has also been updated since that time. Therefore, both the 8th and 12th data items will be invalidated. This method works effectively for clients who have been disconnected for a long time. It optimizes the utilization of bandwidth for invalidation report.

In [Pitoura 1998], a revised version of invalidation reports is designed to provide the semantics of read-only transactions for mobile clients without sending uplink requests to servers.

5. CASE STUDIES

We conclude this review with a study of three prototype systems for mobile information access. These systems serve as a means of demonstrating how new paradigms analyzed in the previous sections are being applied in practice. The three systems, namely, Bayou, Odyssey, and Rover, demonstrate some approaches of new paradigms for mobile client-server computing. Bayou provides a flexible client-server architecture in which a server can be any machine that holds a complete copy of one or more databases. A portable computer may also act as a server for some databases and as a client for others. Odyssey supports application-aware adaptation based on type-specific operations. Rover provides a toolkit for distributed object-based client-server computing. The relocatable objects in Rover enable a flexible client-server architecture for mobile applications. The toolkit supports both application-transparent and application-aware adaptation.

5.1 Bayou

Bayou is a Xerox PARC research project led by Douglas Terry. It is designed for collaborative applications in a mobile computing environment containing portable machines with intermittent network connectivity [Demers et al. 1994; Terry et al. 1994; Terry et al. 1995]. The focus of the Bayou system has been on exploring mechanisms that let mobile clients actively read and write shared data, such as appointment calendars, bibliographic databases, meeting notes, evolving design documents, news bulletin boards etc.

Bayou supports application-specific mechanisms that detect and resolve the update conflicts, ensures that replicas move towards eventual consistency, and defines a protocol by which the resolution of update conflicts stabilizes. Bayou includes consistency management methods for conflict detection, called dependency checks, and per-write conflict resolution based on client-provided merge procedures. To guarantee eventual consistency, Bayou servers are able to rollback the effects of previously executed writes and redo them according to a global serialization order. Furthermore, Bayou permits clients to observe the results of all writes received by a server, including tentative writes whose conflicts have not been ultimately resolved.

5.1.1 *The System Model.* In the Bayou system, each data collection is replicated in full at a number of servers. Applications running as clients interact with the servers through the Bayou application programming interface (API), which is implemented as a client stub bound with the application. This API, as well as the underlying client-server RPC protocol, supports two basic operations: Read and Write. Read operations permit queries over a data collection, while Write operations can insert, modify, and delete a number of data items in a collection. Figure 15 illustrates these components of the Bayou architecture. In the Bayou system, a client and a server may be co-resident on a host, as would be typical of a laptop or PDA running in isolation.

Access to one server is sufficient for a client to perform useful work. The client can read the data held by that server and submit Writes to the server. Once a Write is accepted by a server, the client has no further responsibility for that Write. In particular, the client does not need to wait for the Write to propagate to other servers. In other words, Bayou presents a weakly consistent replication model with a read-any/write-any style of access. While individual Read and Write operations are performed at a single server, clients need not confine themselves to interacting with a single server. In a mobile computing environment, switching between servers

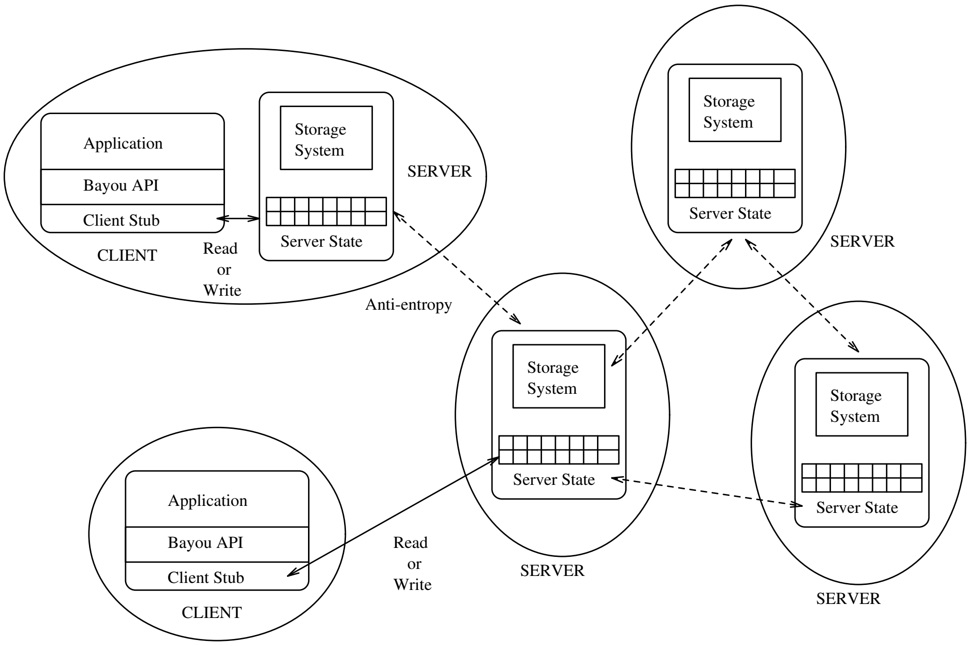

Fig. 15. The Bayou system model.

is often desirable, and Bayou provides session guarantees to reduce client-observed inconsistencies when accessing different servers.

5.1.2 *Bayou Application-Specific Conflict Resolution.* Application-specific conflict detection is adopted in the Bayou system. This approach is motivated by the observation that different applications have different notions of what it means for two updates to conflict, and that such conflicts cannot always be identified by simply observing conventional reads and writes submitted by the applications. In Bayou, storage systems provide means for an application to specify its notion of a conflict along with its policy for resolving conflicts. In return, the system implements mechanisms for reliably detecting conflicts, as specified by the application, and for automatically resolving them when possible.

The Bayou system includes two mechanisms for automatic conflict detection and resolution that are intended to support arbitrary applications: dependency checks and merge procedures. These mechanisms permit clients to indicate, for each individual write operation, how the system should detect conflicts involving the write and what steps should be taken to resolve any detected conflicts based on the semantics of the application.

Application-specific conflict detection is accomplished as follows. Each Write operation includes a dependency check consisting of an application-supplied query and its expected result. A conflict is detected if the query, when run at a server against its current copy of the data, does not return the expected result. This dependency check is a precondition for performing the update that is included in the Write operation. If the check fails, then the requested update is not performed and the server invokes a procedure to resolve the detected conflict.

As an example of application-defined conflicts, consider a Bayou Write operation that might be submitted by the meeting room scheduling application. This Write attempts to reserve an hour-long time slot. It includes a dependency check with a single query that returns information about any previously reserved meetings that overlap with this time slot. It expects the query to return an empty set.

Bayou's dependency checks can also be used to detect Write-Write conflicts. That is, they can be used to detect when two users update the same data item without one of them first observing the other's update. Such conflicts can be detected by having the dependency check query the current values of any data items being updated and ensure that they have not changed from the values they had at the time the Write was submitted.

Bayou's dependency checking mechanism is more powerful than the traditional use of version vectors since it can also be used to detect Read-Write conflicts. Specifically, each Write operation can explicitly specify the expected values of any data items on which the update depends, including data items that have been read but are not being updated. Thus, Bayou clients can emulate the optimistic style of concurrency control employed in some distributed database systems. For example, a Write operation that installs a new program binary file might only include a dependency check of the sources, including version stamps, from which it was derived. Since the binary does not depend on its previous value, this need not be included.

Moreover, because dependency queries can read any data in the server's replica, dependency checks can enforce arbitrary, multi-item integrity constraints on the data. For example, suppose a Write transfers $100 from account A to account B. The application, before issuing the Write, reads the balance of account A and discovers that it currently has $150. Traditional optimistic concurrency control would check that account A still had $150 before performing the requested Write operation. The real requirement, however, is that the account have at least $100, and this can easily be specified in the Write's dependency check. Thus, only if concurrent updates cause the balance in account A to drop below $100 will a conflict be detected.

Once a conflict is detected, a merge procedure is run by the Bayou server in an attempt to resolve the conflict. Merge procedures, included with each Write operation, are general programs written in a high-level, interpreted language. They can have embedded data, such as application-specific knowledge related to the update that was being attempted, and can perform arbitrary Reads on the current state of the server's replica. The merge procedure associated with a Write is responsible for resolving any conflicts detected by its dependency check and for producing a revised update to apply. The complete process of detecting a conflict, running a merge procedure, and applying the revised update is performed atomically at each server as part of executing a Write. Supporting dependency checks separately allows servers to avoid running the merge procedure in the expected case where the Write does not introduce a conflict.

Bayou merge procedures are written by application programmers in the form of templates that are instantiated with the appropriate details filled in for each

Write. The users of applications do not have to know about merge procedures and, therefore, about the internal workings of the applications they use, except when automatic conflict resolution cannot be done. In the case where automatic resolution is not possible, the merge procedure will still run to completion, but it is expected to produce a revised update that logs the detected conflict in a way that will enable a person to resolve the conflict later. To enable manual resolution, the conflicting updates must be presented to a user in a manner that allows him to understand what has happened.

Bayou allows replicas to always remain accessible. This permits clients to continue to Read previously written data and to continue to issue new Writes. In the meeting room scheduling application, for example, a user who only cares about Monday meetings need not concern himself with scheduling conflicts on Wednesday. The potential drawback of this approach is that newly issued Writes may depend on data that is in conflict and may lead to cascaded conflict resolution.

5.1.3 *Bayou Replication Management.* While the replicas held by two servers at any time may vary in their contents because they have received and processed different Writes, a fundamental property of the Bayou design is that all servers move towards eventual consistency. However, it cannot enforce strict bounds on Write propagation delays since these depend on network connectivity factors that are outside of Bayou's control. Two important features of the Bayou system design allow servers to achieve eventual consistency. First, Writes are performed in the same, well-defined order at all servers. Second, the conflict detection and merge procedures are deterministic so that servers resolve the same conflicts in the same manner.

When a Write is accepted by a Bayou server from a client, it is initially deemed tentative. Tentative Writes are ordered according to timestamps assigned to them by their accepting servers. Committed Writes are ordered according to the times at which they commit and before any tentative Writes.

The only requirement placed on timestamps for tentative Writes is that they should be monotonically increasing at each server so that the pair [timestamp, ID of server that assigned it] produce a total order on Write operations. Bayou servers maintain logical clocks to timestamp new Writes. A server's logical clock is generally synchronized with its real-time system clock, but, in order to preserve the causal ordering of Write operations, the server may need to advance its logical clock when Writes are received during anti-entropy.

Enforcing a global order on tentative as well as committed Writes ensures that an isolated cluster of servers will come to agreement on the tentative resolution of any conflicts that they encounter. This is not strictly necessary since clients must be prepared to deal with temporarily inconsistent servers in any case. Moreover, clients can expect that the tentative resolution of conflicts within their cluster will correspond to their eventual permanent resolution, provided that no further conflicts are introduced outside the cluster.

Because servers may receive Writes from clients and from other servers in an order that differs from the required execution order, and because servers immediately apply all known Writes to their replicas, servers must be able to undo the effects of a previous tentative execution of a Write operation and reapply it in a different order. The number of times that a given Write operation is re-executed depends only on the order in which Writes arrive via anti-entropy and not on the likelihood of conflicts involving the Write. Conceptually, each server maintains a log of all Write operations that it has received, sorted by their committed or tentative timestamps, with committed Writes at the head of the log. The server's current data contents are generated by executing all of the Writes in the given order.

Bayou guarantees that merge procedures, which execute independently at each server, produce consistent updates by forcing them to depend only on the server's current data contents and on any data supplied by the merge procedure itself. In particular, a merge procedure cannot access time-varying or server-specific "environment" information such as the current system clock or server's name. Moreover, merge procedures that fail due to exceeding their limits on resource usage must fail deterministically. This means that all servers must place uniform bounds on the CPU and memory resources allocated to a merge procedure and must consistently enforce these bounds during execution. Once these conditions are met, two servers that start with identical replicas will end up with identical replicas after executing a Write.

5.1.4 *Bayou Applications.* The Bayou system is designed to support a variety of non-real-time collaborative applications, such as shared calendars, mail and bibliographic databases, program development, and document editing for disconnected workgroups, as well as applications that might be used by individuals at different hosts at different times.

Meeting room scheduler: The meeting room scheduling application enables users to reserve meeting rooms. At most, one person or group can reserve the room for any given period of time. This meeting room scheduling program is intended for use after a group of people have already decided that they want to meet in a certain room and have determined a set of acceptable times for the meeting. It does not help them to determine a mutually agreeable place and time for the meeting; it only allows them to reserve the room. Thus, it is a much simpler application than one that supports general meeting scheduling.

In this application, Bayou users could interact with a graphical interface for the schedule of a room that indicates which times are already reserved, much like the display of a typical calendar manager. The meeting room scheduling program periodically re-reads the room schedule and refreshes the user's display. This refresh process enables the user to observe new entries added by other users. The user's display might be out-of-date with respect to the confirmed reservations of the room; for example, when it is showing a local copy of the room schedule on a disconnected laptop.

Bayou users reserve a time slot simply by selecting a free time period and filling in a form describing the meeting. Because the user's display might be out-of-date, there is a chance that the user could try to schedule a meeting at a time that was already reserved by someone else. To account for this possibility, users can select several acceptable meeting times rather than just one. At most, one of the requested times will eventually be reserved.

A user's reservation, rather than being immediately confirmed (or rejected), may remain "tentative" for a while. While tentative, a meeting may be rescheduled as other interfering reservations become known. Tentative reservations are indicated as such on the display (by showing them grayed). The "outdatedness" of a calendar does not prevent it from being useful, but simply increases the likelihood that tentative room reservations will be rescheduled and finally "committed" to less preferred meeting times.

A group of Bayou users, although disconnected from the rest of the system, can immediately see each other's tentative room reservations if they are all connected to the same copy of the meeting room schedule. If, instead, users are maintaining private copies on their laptop computers, local communication between the machines will eventually synchronize all copies within the group.

The meeting room scheduling application provides good examples of conflict resolution procedures that are specific not only to a particular application but also to a particular Write operation. In this application, users, well aware that their reservations may be invalidated by other concurrent users, can specify alternate scheduling choices as part of their original scheduling updates. These alternates are encoded in a merge procedure that attempts to reserve one of the alternate meeting times if the original time is found to be in conflict with some other previously scheduled meeting. A different merge procedure altogether could search for the next available time slot to schedule the meeting, which is an option a user might choose if any time would be satisfactory.

Bibliographic database: The application allows users to cooperatively manage databases of bibliographic entries. A user can freely read and write to any copy of the database, such as the one that resides on his laptop. For the most part, the database is append-only, though users occasionally update entries to fix mistakes or add personal annotations.

In bibliographic databases, each entry has a unique, human-sensible key that is constructed by appending the year in which the paper was published to the first author's last name and adding a character if necessary to distinguish between multiple papers by the same author in the same year. Thus, the first paper by Jones et al. in 1995 might be identified as "Jones95" and subsequent papers as "Jones95b", "Jones95c", and so on.

An entry's key is tentatively assigned when the entry is added. A user must be aware that the assigned keys are only tentative and may change when the entry is "committed." In other words, a user must be aware that other concurrent updaters could be trying to assign the same key to different entries. Only one entry can have the key; the others will be assigned alternative keys by the system. Thus, for example, if the user employs the tentatively assigned key in some fashion, such as embedding it as a citation in a document, then he must also remember to check later that the key assigned when the entry was committed is, in fact, the expected one.

Because users can access inconsistent database copies, the same bibliographic entry may be concurrently added by different users with different keys. To the extent possible, the system detects duplicates and merges their contents into a single entry with a single key. In this application, a Bayou user may choose to operate in disconnected mode even if constant connectivity were possible. For example, a Bayou user may be in a university library looking up papers. He occasionally types bibliographic references into his laptop or PDA.

5.2 Odyssey

Odyssey is a CMU research project led by M. Satyanarayanan. It addresses an application-aware adaptation approach to deal with application diversity and concurrency in mobile environments. The application-aware adaptation is implemented with the support of system-coordinated type-specific operations. It supports concurrent execution of diverse mobile applications that execute on mobile clients, but read or update remote data on servers [Kumar and Satyanarayanan 1993; Satyanarayanan et al. 1994; Noble, Price, and Satyanarayanan 1995; Noble et al. 1997].

5.2.1 *Odyssey Client Architecture.* In Odyssey, the data accessed by an application may be stored in one or more general-purpose repositories such as file servers, SQL servers, or Web servers. It may also be stored in more specialized repositories such as video libraries, query-by-image-content databases, or back–ends of geographical information systems.

Ideally, a data item available on a mobile client should be indistinguishable from that available to the accessing application if it were to be executed on the server storing that item. But this correspondence may be difficult to preserve as mobile resources become scarce; some form of degradation may be inevitable. In Odyssey, *fidelity* is used to describe the degree to which data presented at a client matches the reference copy at the server. Fidelity has many dimensions. One well-known, universal dimension is consistency. For Video applications, data has at least two additional dimensions: frame rate and image quality of individual frames. Odyssey provides a framework within which diverse notions of fidelity can be incorporated.

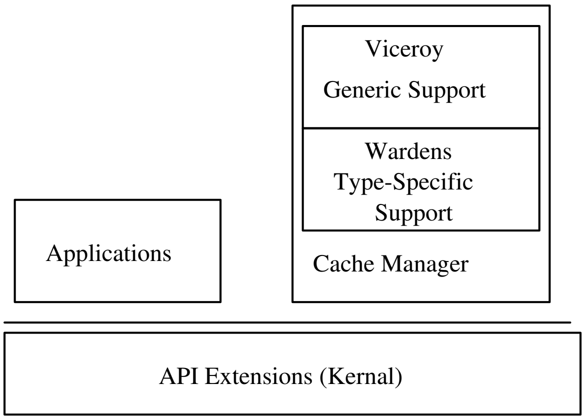

Fig. 16. Odyssey client architecture

Odyssey implements an approach of application-aware adaptation. The system monitors resource levels, notifies applications of relevant changes, and enforces resource allocation decisions. Each application independently decides how best to adapt when notified. Odyssey incorporates type-awareness via specialized code components called wardens. A warden encapsulates the system-level support at a client necessary to effectively manage a data type. To fully support a new data type, an appropriate warden has to be written and incorporated into Odyssey at each client. The wardens are subordinate to a type-independent component called the viceroy, which is responsible for centralized resource management (see Figure 16). The collaborative relationship in the application-aware adaptation is realized in two parts. The first, between the viceroy and its wardens, is data-centric: it defines the fidelity levels for each data type and factors them into resource management. The second, between applications and Odyssey, is action-centric: it provides applications with control over the selection of fidelity levels supported by the wardens.

5.2.2 *Odyssey System Components.* Odyssey supports application-aware adaptation by enabling an application to

—operate on Odyssey objects,

—express resource expectations,

—be notified when expectations are no longer met, and

—respond by changing fidelity.

Operating on Odyssey Objects: Odyssey is integrated into NetBSD as a new VFS file system [Noble et al. 1997]. As shown in Figure 16, the viceroy and wardens are implemented in user space rather than in the kernel. Operations on Odyssey objects are redirected to the viceroy by a small in-kernel interceptor module. All other system calls are handled directly by NetBSD.

Wardens are statically linked with the viceroy, and the ensemble executes in a single address space with user-level threads. Communication between the viceroy and wardens is through procedure calls and shared data structures. The wardens are entirely responsible for communicating with servers and caching data from them when appropriate; applications never contact servers directly.

Expressing Resource Expectations: Applications communicate resource expectations to Odyssey using the request system call shown in Figure 17(a). The call takes a resource descriptor identifying a resource and specifying a window of tolerance based upon availability. This call expresses the application's desire to be told if the availability of the resource strays outside the window. If, at the time of the request, the availability of the resource is within the window of tolerance, the viceroy registers the request and returns a unique identifier for it. This identifier can be used by the viceroy in notifying the application that the resource has left the requested bounds or by the application in a future cancel system call to discard the registered request.

If the resource is currently outside the bounds of the tolerance window, an error code and the current available resource level are returned. The application is then expected to try again, but this time with a new window of tolerance that corresponds to a new fidelity level. The fields of a resource descriptor are shown in Figure 17(b). Each resource is named by a unique resource identifier. Figure 17(c) lists the generic resources that Odyssey manages. The most critical resource in mobile computing is the network bandwidth. The window of tolerance is indicated by lower and upper bounds. A resource descriptor also specifies the name of a procedure that will be called to notify the application that the resource has left the window.

request(in path, in resource-descriptor, out request-id)
cancel(in request-id)

(a) Resource Negotiation Operations

resource-id
lower bound
upper bound
name of upcall handler

(b) Resource Descriptor Fields

Network Bandwidth	bytes/second
Network Latency	microseconds
Disk Cache Space	kilobytes
CPU	SPECCint95
Battery Power	minutes
Money	cents

(c) Generaic Resource in Odyssey

handler(in request-id, in resource-id, id resource-level)

(d) Upcall Handler

tsop(in path, in opcode, in insize, in inbuf,
inout outsize, out outbuf)

(e) Type-Specific Operations

Fig. 17. Odyssey API

Notifying Applications: When the viceroy discovers that the availability of a resource has strayed outside a registered window of tolerance, it generates an upcall to the corresponding application. The application adjusts its fidelity according to its individual policy. It then issues another request call to register a revised window of tolerance appropriate to the new fidelity.

An upcall handler is invoked with three parameters, as shown in Figure 17(d). The first parameter identifies the request operation on whose behalf the upcall is being delivered. The second parameter identifies the resource whose availability has changed, and the third parameter gives the new availability. Upcalls closely resemble UNIX signals, but offer improved functionality. Like signals, upcalls can be sent to one or more processes, can be blocked or ignored, and have similar inheritance semantics on the process fork. Unlike signals, upcalls offer semantics in a specific order only once for each receiver of a particular upcall. Further, upcalls allow parameters to be passed to target processes and results to be returned.

Changing Fidelity: Requests for fidelity changes do not map well to the NetBSD file system interface. Further, many data types have natural access methods that are not well supported by the untyped byte stream model. To address these shortcomings, a general escape purpose mechanism called tsop, or type-specific operation, is included shown in Figure 17(e). The arguments to tsop specify an Odyssey object and the opcode of a type-specific operation to be performed on it. Input and output parameters are specified as unstructured memory buffers, in the spirit of the *ioctl* system call.

5.2.3 *Odyssey Applications.* Several applications are modified to demonstrate Odyssey's ability to support application diversity. Two of them are drawn from the domain of mobile information access: a video player and a Web browser. Each application requires a different strategy for integration into Odyssey, and each has distinct notions of fidelity.

Video Player: Video Player is based on *xanim*, a public-domain software package that can generate video animation from data stored in various formats in a local file. A warden is written to satisfy client requests and fetch data from the server, as shown in Figure 18.

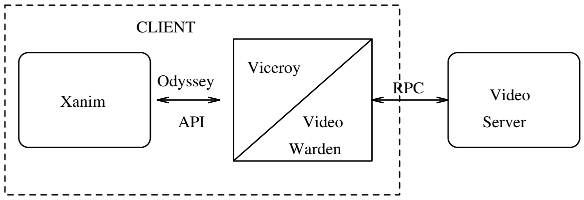

Fig. 18. Video player in Odyssey

Each movie is stored in multiple tracks at the server, one track per fidelity level. Three levels of fidelity for Quicktime video data are incorporated: JPEG-compressed color frames at qualities 99 and 50, and black-and-white frames. The warden supports two tsops: to read a movie's meta-data and to get a particular frame from a specified track. The warden performs read-ahead of frames to lower latency. When the player opens a movie, it calculates the bandwidth requirements of each track from the movie meta-data. The player begins the movie at highest possible quality and registers the corresponding window of tolerance with Odyssey. When it is notified of a significant change in bandwidth, the player determines a new fidelity level and switches to the corresponding track. If the player switches from a low fidelity track to a higher one, the warden discards the prefetched low-quality frames.

Web Browser:

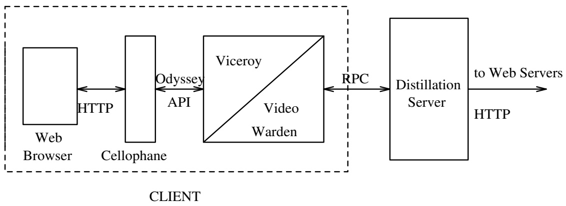

Fig. 19. Web browser in Odyssey

Netscape browser's proxy facility is exploited to take advantage of Odyssey, as shown in Figure 19. All of Netscape's requests are redirected to a client module called the cellophane. Together, Netscape and the cellophane constitute a single application from the viewpoint of Odyssey. The cellophane makes use of the Odyssey API and selects fidelity levels. Netscape passively benefits from the adaptation initiated by the cellophane. The cellophane transforms HTTP requests from Netscape into file operations on Odyssey Web objects. The Web warden forwards these requests via the client's mobile network connection to a distillation server. At the highest fidelity, images are uncompressed. Progressively lower levels correspond to JPEG-compressed images of decreasing quality. The warden provides a tsop to set the fidelity level. The distillation server fetches requested objects from the appropriate Web server, distills them to the requested fidelity level, and sends the results to the warden. The data is then passed to Netscape via the cellophane. These steps are completely transparent to both Netscape and the Web server; each perceives normal Web access.

5.3 Rover

Rover is a research project at MIT led by M. Kaashoek. The Rover toolkit offers an environment to support both application-transparent and application-aware adaptation for mobile client-server applications [Joseph, Tauber, and Kaashoek 1996; Joseph, Tauber, and Kaashoek 1997]. The application-transparent adaptation is realized by developing proxies for system services that hide the mobile characteristics of the environment from applications. The application-aware adaptation is supported by the use of relocatable dynamic objects in the construction of client and server applications. The Rover toolkit provides a framework to construct mobile applications with flexible client-server architecture.

5.3.1 *Rover Toolkits.* The Rover Toolkit offers applications a distributed object system based on a client-server architecture [Joseph, Tauber, and Kaashoek 1997] (see Figure 20). Clients are Rover applications that typically run on mobile hosts, but can also run on stationary hosts as well. Servers, which may be replicated, typically run on stationary hosts and hold the long-term state of the system. Communication between clients is limited to peer-to-peer interactions within a mobile host (using the local object cache for sharing) and mobile host–server interactions; there is no support for peer-to-peer, mobile host to mobile host interactions. The Rover toolkit provides mobile communication support based on two ideas: relocatable dynamic object (RDO) and queued remote procedure call (QRPC). A relocatable dynamic object is an object (code and data) with a well-defined interface that can be dynamically loaded into a client computer from a server computer, or vice versa, to reduce client-server communication requirements. Queued remote procedure call is a communication system that permits applications to continue to make non-blocking remote procedure calls even when a host is disconnected — requests and responses are exchanged upon network reconnection.

Rover gives applications control over the location where the computation will be performed. In an intermittently connected environment, the network often separates an application from the data upon which it is dependent. By moving RDOs across the network, applications can move data and/or computation from the client to the server and vice versa.

Use of RDOs allows mobile-aware applications to migrate functionality dynam-

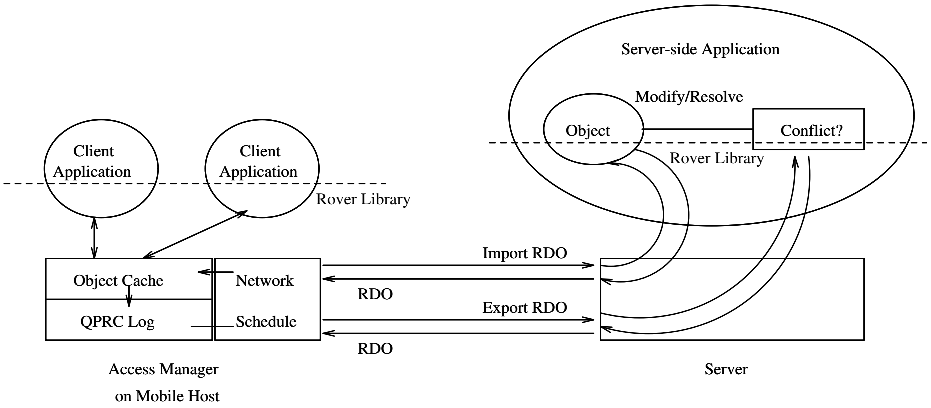

Fig. 20. Rover's relocatable dynamic object (RDO) architecture

ically to either side of a slow network connection to minimize the amount of data communicated across the network. Caching RDOs reduces latency and bandwidth consumption. Interface functionality can run at full speed on a mobile host while large data manipulations may be performed on the well-connected server. All application code and all application-touched data are written as RDOs. RDOs may execute at either the client or the server. Each RDO has a "home" server that maintains the primary, canonical copy. Clients import secondary copies of RDOs into their local caches and export tentatively updated RDOs back to their home servers. RDOs may vary in complexity from simple calendar items with a small set of operations to modules that encapsulate a significant part of an application (e.g., the graphical user interface for an email browser). Complex RDOs may create a thread of control when they are imported.

Rover clients use QRPC to lazily fetch RDOs from servers (see Figure 20). When an application issues a QRPC, Rover stores the QRPC in a local stable log and immediately returns control to the application. If the application has registered a callback routine, then when the requested RDO has arrived, Rover will invoke the callback to notify the application. Alternatively, applications may simply block to wait for critical data (although this is an undesirable action, especially when the mobile host is disconnected). When the mobile host is connected, the Rover network scheduler drains the log in the background, forwarding any queued QRPCs to the server.

When a Rover application modifies a locally cached RDO, the cached copy is marked tentatively committed. Updates are committed by using QRPC to lazily propagate the mutating operations to the Rover server, where they are applied to the canonical copies. In the meantime, the application may choose to use tentatively committed RDOs. This allows the application to continue execution even if the mobile host is disconnected.

As shown in Figure 21, the Rover Toolkit consists of four key components: the access manager, the object cache (client-side only), the operation log, and the network scheduler. Each machine has a local Rover access manager which is responsible for

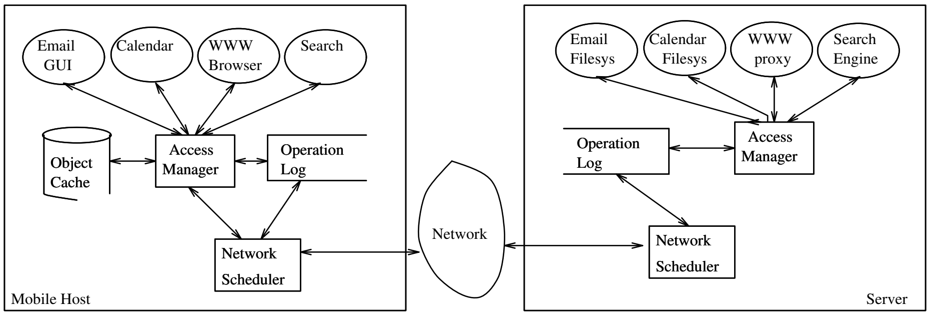

Fig. 21. Rover component architecture

handling all interactions between client-side and server-side applications and among client-side applications. The access manager services requests for objects (RDOs), mediates network access, logs modifications to objects, and, on the clients' side, manages the object cache. Client-side applications communicate with the access manager to import objects from servers and cache them locally. Server-side applications are invoked by the access manager to handle requests from client-side applications. Applications invoke the methods provided by the objects and, using the access manager, make the changes globally visible by exporting them back to the servers.

Within the access manager, RDOs are imported into the object cache while QRPCs are exported to the operation log. The access manager routes invocations and responses between applications, the cache, and the operation log. The log is drained by the network scheduler, which mediates between the various communication protocols and network interfaces.

The object cache provides stable storage for local copies of imported objects. The object cache consists of a local private cache located within the application's address space and a global shared cache located within the access manager's address space. Client-side applications do not usually interact directly with the object cache. When a client-side application issues an import or export operation, the Toolkit satisfies the request depending on whether the object is found in a local cache and on the consistency option specified for the object.

Once an object has been imported into the client-side application's local address space, method invocations without side effects are serviced locally by the object. At the application's discretion, method invocations with side effects may also be processed locally, inserting tentative data into the object cache. Operations with side effects also insert a QRPC into a stable operation log located at the client. Each insert is a synchronous action. Support for intermittent network connectivity is accomplished by allowing the log to be incrementally flushed back to the server. Thus, as network connectivity comes and goes, the client will make progress towards reaching a consistent state.

The network scheduler contributes to log transmission optimization by grouping operations destined to the same server for transmission and selecting the appropriate transport protocol and medium over which to send them. Rover is capable of using a variety of network transports. Rover supports both connection-based protocols (e.g., HTTP over TCP/IP networks) and connection-less protocols (e.g., SMTP over IP or non-IP networks). The network scheduler leverages the queuing of QRPCs performed by the log to gain transmission efficiency.

5.3.2 *Constructing Mobile Applications Using Rover.* There are several steps involved in porting an existing application to Rover or creating a new Rover-based application. Each step requires the application developer to make several implementation choices. The first step is to split the application into components and identify which components should be present on each side of the network link. The division will be mostly static because most of the file system components will remain on the server and most of the GUI components will remain on the client. However, those components that are dependent upon the computing environment (network or computational resources) or are infrequently used may be dynamically relocated.

Once the application has been split into components, the next step is to appropriately encapsulate the application's state within objects that can be replicated and sent to multiple clients. In migrating to the mobile environment, the application's reading of files is replaced by importing of objects, and its writing of files is replaced by exporting of changes to objects. The file system interface still exists in the server-side of the application. However, inserted between the two halves of the application is an object layer.

One of the primary purposes of the object layer is to provide a means of reducing the number of network messages that must be sent between the client and the server; this is done by migrating the computation.

The next step is to add support for interacting with the environment. For example, in an e-mail message, one of the important pieces of message meta-data that a folder object contains is the message size and the size of any attachments. This information can be used by the application and conveyed to the user to allow useful decisions to be made. Support for prefetching is another environment interaction issue. Also, the application developer must decide which mechanisms to use for notifying users of the status of displayed data.

The final and important step is the addition of application-specific conflict resolution procedures. For most stationary environments, conflicts are infrequent. For the mobile environment, they will be more common. Application developers can leverage the additional semantic information that is available with Rover's operation-based (instead of value-based) approach to object updating.

The programming interface between Rover and its client applications contains four primary functions: *create session*, *import*, *export*, and *invoke*. Client applications call create session once with authentication information to set up a connection with the local access manager and receive a session identifier. The authentication information is used by the access manager to authenticate the client requests sent to Rover servers.

To import an object, an application calls import and provides the object's unique identifier, the session identifier, a callback, and arguments. In addition, the application specifies a priority that is used by the network scheduler to reorder QRPCs. The import function immediately returns a promise to the application. The application can then wait on this promise or continue execution. Rover transparently queues QRPCs for each import operation in the stable log. When the requested object is received by the access manager, the access manager updates the promise with the returned information. In addition, if a callback was specified, the access manager invokes it.

Once an object is imported, an application can invoke methods on it to read and/or change it. Applications export each local change to an object back to the servers by calling the export operation and providing the object's unique identifier, the session identifier, a callback, and arguments. Like import, export immediately returns a promise. When the access manager receives responses to export, it updates the affected promises and invokes any application-specified callbacks.

5.3.3 *Rover Applications.* Two mobile-transparent applications that have been implemented using the Rover Toolkit are Rover NNTP proxy, a USENET reader proxy, and Rover HTTP proxy, a proxy for Web browsers. Mobile-aware applications implemented using Rover are Rover Exmh, an e-mail browser; Rover Webcal, a distributed calendar tool; Rover Irolo, a graphical rolodex tool; and Rover Stock Market Watcher, a tool that obtains stock quotes.

Two of the mobile-aware applications are based upon existing UNIX applications. Rover Exmh is a port of Brent Welch's Exmh Tcl/Tk-based e-mail browser. Rover Webcal is a port of Ical, a Tcl/Tk and C++ based distributed calendar and scheduling program written by Sanjay Ghemawat. Rover Irolo and the Rover Stock Market Watcher were built from scratch.

The Rover HTTP and NNTP proxies demonstrate how Rover mobile-aware proxies support existing applications (e.g., Netscape and xrn) without modification (i.e., mobile-transparent applications).

Rover NNTP proxy: Using the Rover NNTP proxy, users can read USENET news with standard news readers while disconnected and receive news updates even over very slow links. Whereas most NNTP servers download and store all available news, the Rover proxy cache is filled on a demand-driven basis. When a user begins reading a newsgroup, the NNTP proxy loads the headers for that newsgroup as a single RDO while articles are prefetched in the background. As the user's news reader requests the header of each article, the NNTP proxy provides them by using the local newsgroup RDO. As new articles arrive at the server, the server-side of the proxy constructs operations to update the newsgroup-header object. Thus, when a news reader performs the common operation of rereading the headers in a newsgroup, the NNTP proxy can service the request with minimal communication over the slow link.

Rover HTTP proxy: This application allows users of existing Web browsers to "click ahead" of the arrived data by requesting multiple new documents before earlier requests have been satisfied. The proxy intercepts all web requests and, if the requested item is not locally cached, returns a null response to the browser and enqueues the request in the operation log. When a connection becomes available, the page is automatically requested. In the meantime, the user can continue to browse already available pages and issue additional requests for pages without waiting. The granularity of RDOs is individual pages and images. The client and server cooperate in prefetching. The client specifies the depth of prefetching for pages while the server automatically prefetches in-lined images. The proxy uses a separate window (from the browser) to display the status of a page (loaded or pending). If an uncached file is requested and the network is unavailable, an entry is added to the window. As pages arrive, the window is updated to reflect the changes. This window exposes the object cache and operations log directly to the user and allows the user limited control over them.

Rover Exmh: Rover Exmh uses three types of RDOs: mail messages, mail folders, and lists of mail folders. By using this level of granularity, many user requests can be handled locally without any network traffic. Upon startup, Rover Exmh prefetches the list of mail folders, the mail folders the user has recently visited, and the messages in the user's inbox folder. Alternatively, using a finer-level granularity (e.g., header and message body) would allow for more prefetching, but could delay servicing of user requests (especially during periods of disconnection). Using a larger granularity (e.g., entire folders) would seriously affect usability and response time for slow links.

Some computation can be migrated to servers. For example, instead of performing a glimpse search of mail folders locally at the client (and thus having to import the index across a potentially low bandwidth link), the client can construct a query request RDO and send it to the server.

The GUI indicates that an operation is tentative through color coding. Conflict detection is based upon a log of changes to RDOs; this allows the server to detect and resolve a conflict, such as one user adding a message to a folder and another user deleting it. Unresolvable conflicts are reflected back to the user.

Rover Webcal: This distributed calendar tool uses two types of RDOs: items (appointments, daily to do lists, and daily reminders) and calendars (lists of items). At this level of granularity, the client can fetch calendars and then prefetch items using a variety of strategies (e.g., plus or minus one week, a month at a time, etc.).

Rover Webcal uses color coding to aid the user in identifying those objects that have been locally modified but not yet propagated to a server. Conflict detection is based upon a log of changes to RDOs; this allows the server to detect and resolve a conflict, such as one user adding an item to a calendar and another user deleting it.

Rover Irolo: This graphical rolodex application uses two types of RDOs: entries and indices (lists of entries). The GUI displays the last time an entry was updated and indicates whether the item is committed or tentative. Conflict detection is based upon a log of changes to RDOs; this allows the server to detect and resolve a conflict such as one user adding an entry to an index and another user deleting it.

Rover Stock Market Watcher: This application uses both computation migration and fault-tolerance techniques. The client constructs RDOs for stocks that are to be monitored and sends them to the server. The server uses fault-tolerant techniques to store the real-time information retrieved from stock ticker services.

5.4 Summary

Tables 1, 2, and 3 summarize the three systems, Bayou, Odyssey, and Rover, used as case studies in this section. The tables highlight the applications each of these systems supports, and the mobile client-server computing strategies and techniques each of them has implemented.

System	Bayou
Applications	non-real-time collaborative applications: meeting room scheduler and bibliographic database, appointment calendars, evolving design documents, news bulletin boards
Adaptation	application-specific adaptation to disconnection & intermittent connectivity; applications are permitted to make trade-off of replicated data consistency & availability by using individually selectable session guarantees
Model	collaborative and flexible group-based client-server Architecture and full (or disconnected) client architecture
Mobile Data	system support for detection of update conflicts, application-specific resolution of update conflicts, eventual replica convergence through a peer-wise anti-entropy process, and per-client consistency guarantees

Table 1. Summary of the Bayou system

System	Odyssey
Applications	file system access, video playing, and Web browsing
Adaptation	client-based application adaptation with the system support that provides resource monitoring, notifies applications of resource changes, and enforces resource allocation decisions
Model	classic client-server architecture
Mobile Data	client pull based data (distilled copies) delivery

Table 2. Summary of the Odyssey system

System	Rover
Applications	e-mail, appointments, todo lists, reminders, calendars; Web pages and images
Adaptation	client-server application adaptation with the use of Rover Toolkits that deal with intermittent and low-bandwidth connection and resource-poor clients; application-transparent adaptation by using Rover proxies
Model	flexible client-server architecture with the use of RDOs
Mobile Data	asynchronous RDOs pull (import) and push (export)

Table 3. Summary of the Rover system

6. CONCLUSION

With advances in wireless data telecommunications and portable computers, nomadic users will soon enjoy virtually unlimited access to information and services any time and anywhere. There are, however, obstacles and limitations inherent in the wireless environment. The unpredictable mobility of the user adds a great deal of complexity to the problem. Such obstacles render the traditional approach to designing and implementing client-server applications insufficient in meeting nomadic users' expectations.

Using assumptions linked to second generation wireless networks and to less than ideal mobile computers (general purpose portable computers with limited resources), researchers realized that either application or system adaptation is a key requirement in any mobile computing system. Researchers also realized that they must identify and systematically eliminate the limitations of this new environment through various optimizations.

This survey explores issues related to the development and deployment of such adaptive mobile applications. In particular, it focuses on new designs and computing paradigms for mobile client-server applications. It also reviews system architecture that supports such applications. The survey identifies key characteristics that distinguish mobile client-server computing from its traditional counterpart. The survey also examines how these computing characteristics impact information services and applications and discusses new computing paradigms that are required to deal with these impacts. One contribution of this survey is a categorization of emerging computing paradigms for mobile client-server applications. The *mobile-aware adaptation*, *extended client-server model*, and *mobile data access* are three categories that are explained and analyzed. For each category, examples that show the related design and implementation issues are detailed. The survey also provides a comparative review of three major and ongoing research prototypes of mobile client-server information systems, namely, Bayou from Xerox Labs, Odyssey from CMU, and Rover from MIT. These prototypes demonstrate the applications of new computing paradigms in building adaptive mobile client-server systems and applications.

Mobile computing (including mobile client-server information access) is a rapidly changing research field that depends on a rapidly evolving set of technologies. The advent of the so called third generation wireless communication systems, such as UMTS, is an indication of the importance of research in this area. Researchers, however, should monitor these advances closely and should adapt and direct their research based on the parameters of the latest technology. This is important because some of the strong assumptions regarding the limitations of the wireless network or the mobile computer are being relaxed or nullified by newer technological developments.

Many problems still remain to be understood and solved. At this time, it is intricately difficult to quantitatively evaluate and compare the various proposed models and techniques because of the lack of available application experiences and samples. Experimentation with these models should be the critical next step in mobile computing research. We hope that this comprehensive review will help enhance the understanding of the opportunities and limitations of mobile client–server computing. We also hope that the review substantiates the importance of recognizing and understanding emerging technologies in shaping the future directions of research in this area.

REFERENCES

Acharya, S., Alonso, R., Franklin, M., and Zdonik, S. 1995. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, San Jose, California.

Acharya, S., Franklin, M., and Zdonik, S. 1997. Balancing push and pull for data broadcast. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 183–194.

Brooks, C., Mazer, M., Meeks, S., and Miller, J. 1996. Application-specific Proxy Servers as HTTP Stream Transducers. In *Proceedings of the Fourth International World Wide Web Conference (WWW-4)*.

Bartlett, J. 1994. W4-the wireless world wide web.

Bharghavan, V. and Gupta, V. 1997. A framework for application adaptation in mobile computing environments. In *Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC)*, 573–579.

Barbara, D., and Imielinski, T. 1994. Sleepers and workaholics: Caching strategies for mobile environments. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1–12.

Brewer, E., Katz, R., Chawathe, Y., Gribble, S., Hodes, T., Nguyen, G., Stemm, M., Henderson, T., Amir, E., Balakrishnan, H., Fox, A., Padmanabhan, V., and Seshan, S. 1998. A network architecture for heterogeneous mobile computing. *IEEE Personal Communications*, 5(5):8–24.

CITRIX. http://www.citrix.com

CompactHTML. 1998. http://www.w3.org/submission/1998/04/

Chang, H., Tait, C., Cohen, N., Shapiro, M., Mastrianni, S., Floyd, R., Housel, B., and Lindquist, D. 1997. Web browsing in a wireless environment: Disconnected and asynchronous operation in artour web express. In *Proceedings of the Third ACM/IEEE International Conference on Mobile Computing and Networking.*

Davis, N., Friday, A., Wade, S. and Blair, G. 1998. L2imbo: A Distributed Systems Platform for Mobile Computing. *ACM Mobile Networks and Applications (MONET)*, 3(2):143–156.

Duran, J. and Laubach, A. 1999. Virtual personal computers and the portable network. In *Proceedings of the IEEE International Performance, Communication, and Computing Conference*, Phoenix, Arizona.

Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M., and Welch, B. 1994. The bayou architecture: Support for data sharing among mobile users. In *Proceedings of the First Workshop on Mobile Computing Systems and Applications (WMCSA).*

Elmagarmid, A. , Jing, J., and Bukhres, O. 1995. An efficient and reliable reservation algorithm for mobile transactions. In *Proceedings of the International Conference on Information and Knowledge Management*, Baltimore, Maryland, USA.

Fox, A., Gribble, S., Brewer, E., and Amir, E. 1996. Adapting to network and client variation via on-demand, dynamic distillation. In *Proceedings of the 7th Architectural Support for Programming Languages and Operating Systems (SPLOS).*

Gray, R. 1995. AgentTcl: A Transportable Agent System. In *Proceedings of the CIKM Workshop on Intelligent Information Agents.*

General Magic Inc. 1997. *Odyssey Product Information.* http://www.genmagic.com/agents/odyssey.html

HDML. 1997. http://www.uplanet.com

Herman, G., Gopal, G., Lee, K., and Weinrib, A. 1987. Datacycle architecture for very high throughput database systems. In *Proceedings of the ACM SIGMOD Conference on*

Management of Data.

HONEYMAN, P., HUSTON, L., REES, J., AND BACHMANN, D. 1992. *The LITTLE WORK Project.* In *Proceedings of the Third Workshop on Workstation Operating Systems*, Key Biscayne, Florida.

HOUSEL, B. AND LINDQUIST, D. 1996. Webexpress: A system for optimizing web browsing in a wireless environment. In *Proceedings of the Second ACM/IEEE International Conference on Mobile Computing and Networking.*

HEIDEMANN, J., PAGE, T., GUY, R., AND POPEK, G. 1992. Primarily disconnected operation: Experience with Ficus. In *Proceedings of the Second Workshop on the Management of Replicated Data.*

IBM TOKYO RESEARCH LABS 1997. *The Aglet Workbench: Programming Mobile Agents in Java.* http://www.trl.ibm.co.jp/aglets/

IMIELINSKI, T. AND BADRINATH, B. 1994. Wireless mobile computing: Challenges in data management. *Communications of the ACM*, 37(10).

IMIELINSKI, T., VISHWANATH, S., AND BADRINATH, B. 1994. Energy efficient indexing on air. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Minneapolis, Minnesota.

IMIELINSKI, T., VISHWANATH, S., AND BADRINATH, B. 1994. Power efficient filtering of data on the air. In *Proceedings of the International Conference of EDBT (Extending DataBase Technology).*

JING, J., ELMAGARMID, A., HELAL, A., AND ALONSO, R. 1997. Bit-sequences: An adaptive cache invalidation method in mobile client/server environments. *ACM/Baltzer Journal of Mobile Networks and Applications*, 2(2):115–127.

JAIN, R. AND KRISHNAKUMAR, N. 1994. Service handoffs and virtual mobility for delivery of personal information services to mobile users. Technical Report TM-24696, Bellcore.

JOSEPH, A., TAUBER, J., AND KAASHOEK, M. 1996. Building reliable mobile-aware applications using the rover toolkit. In *Proceedings of the Second ACM/IEEE International Conference on Mobile Computing and Networking.*

JOSEPH, A., TAUBER, J., AND KAASHOEK, M. 1997. Mobile computing with the rover toolkit. *IEEE Transactions on Computers: Special issue on Mobile Computing*, 43(3).

KATZ, R. 1994. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1:6–17.

KRISHNAKUMAR, N. AND JAIN, R. 1994. Protocols for maintaining inventory databases and user service profiles in mobile sales applications. In *Proceedings of the Mobidata Workshop*, Rutgers University.

KUENNING, G. AND POPEK, G. 1997. Automated Hoarding for Mobile Computers. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP).*

KOJO, M., RAATIKAINEN, K., AND ALANKO, T. 1994. Connecting mobile workstations to the internet over a digital cellular telephone network. In *Proceedings of the Mobidata Workshop.*

KISTLER, J. AND SATYANARAYANAN, S. 1992. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10:3–25.

KUMAR, P. AND SATYANARAYANAN, M. 1993. Supporting application-specific resolution in an optimistically replicated file system. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, Napa, California.

MUMMERT, L. AND SATYANARAYANAN, M. 1994. Large granularity cache coherence in the coda file system. In *Proceedings of the USENIX Summer Conference*, Boston, MA.

NARAYANASWAMY, S., ET AL. 1996. Application and network support for InfoPad. *IEEE Personal Communications*, 3(2).

NOBLE, B., PRICE, M., AND SATYANARAYANAN, M. 1995. A programming interface for application-aware adaptation in mobile computing. In *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing.*

NOBLE, B., SATYANARAYANAN, S., NARAYANAN, D., TILTON, J., FLINN, J., AND WALKER, K. 1997. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles.*

Pitoura, E. 1998. Supporting Read-Only Transactions in Wireless Broadcasting. In *Proceedings of the DEXA98 International Workshop on Mobility in Databases and Distributed Systems*, 428–433.

Satyanarayanan, M. 1996. Mobile information access. *IEEE Personal Communications*, 3(1).

Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E., and Steere, D. 1990. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39:447–459.

Seshan, S., Le, M., Burgard, F., and Rabaey, J. 1993. Software architecture of the infopad system. In *Proceedings of the Mobidata Workshop.*

Satyanarayanan, M., Noble, B., Kumar, P., and Price, M. 1994. Application-aware adaptation for mobile computing. In *Proceedings of the 6th ACM SIGOPS European Workshop.*

Su, C. and Tassiulas, L. 1998. Joint broadcast scheduling and user's cache management for efficient information delivery. In *Proceedings of the fourth annual ACM/IEEE international conference on mobile computing and networking (MobiCom)*, 33–42.

Tait, C. and Duchamp, D. 1991. Service interface and replica consistency algorithm for mobile file system clients. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, 190–197.

Tait, C. and Duchamp, D. 1992. *An Efficient Variable-Consistency Replicated File Service.* Technical Report. Department of Computer Science, Columbia University, New York.

Terry, D., Demers, A., Petersen, K., Spreitzer, M., Theimer, M., and Welch, B. 1994. Session guarantees for weakly consistent replicated data. In *Proceedings International Conference on Parallel and Distributed Information Systems (PDIS).*

Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., and Hauser, C. 1995. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP-15).*

Vishwanath, S. and Imielinski, T. 1995. Pyramid broadcasting for video on demand service. In *Proceedings of the IEEE Multimedia Computing and Networks Conference*, San Jose, California.

WAP. http://www.wapforum.org

Welling, G. and Badrinath, B. 1998. An architecture for exporting environment awareness to mobile computing applications. *IEEE Transactions on Software Engineering*, 24(5):391–400.

Zenel, B. and Duchamp, D. 1997. General purpose proxies: solved and unsolved problems. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, 87–92.