Rover: A Toolkit for Mobile Information Access

Anthony D. Joseph, Joshua A. Tauber, Alan deLespinasse*
{adj,josh,aldel}@lcs.mit.edu
May 12, 1995

1 Introduction

Application architects for mobile "roving" computers face a unique set of communication and power constraints that are not present in traditional workstation settings. For example, although mobile communication infrastructures are becoming more common, in mobile environments network bandwidth will often be limited, and at times, unavailable. Thus mobile application architects have a common need for system facilities that minimize their dependence on continuous connectivity and that provide tools to optimize their utilization of network bandwidth.

The Rover toolkit was developed to provide mobile application developers with a set of tools to isolate mobile applications from the limitations of mobile communication systems. The Rover toolkit provides mobile communication support based on two ideas: relocatable dynamic objects and queued remote procedure call (QRPC). A relocatable dynamic object is a program with a defined interface that can be dynamically loaded into a client computer from a server computer to reduce client-server communication requirements. Queued remote procedure call is a communication system that permits applications to continue to make RPC requests even when a host is disconnected, with requests and responses being exchanged upon network reconnection.

The Rover implementation is fully compatible with HTTP, and permits applications full access to the resources of the WWW. We have developed several applications, in which dynamic objects can be downloaded into the client and executed to reduce communication requirements, and server requests can be queued for later processing.

Experience with Rover-based applications shows that it is possible to build interactive applications that can isolate a user from loss of network connectivity and from limited network bandwidth. Certain aspects of Rover are also attractive for workstation applications. For example, Rover can send RPCs to remote hosts using electronic mail as a transport, which can permit servers to batch process requests, and can shift low priority inquiries to off hours. Responses to these RPCs are transported by SMTP to a nearby mail host, which has the benefit of staging the responses for rapid client retrieval.

2 Programming Model

Rover provides a traditional object based client-sever programming model that is extended with dynamic relocatable objects and queued remote procedure call. Rover applications run on mobile hosts and *import* objects into their address spaces, *invoke* methods provided by the objects, and *export* the results back to servers. This process is reflected in the first three steps listed in Figure 1.

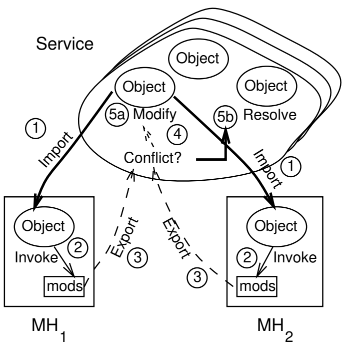

Figure 1: The Rover object model

Imported objects are secondary copies of objects, with the exporting server retaining the primary copy. Programmers manage copy consistency using Rover-provided tools. Rover applications can use callback procedures to isolate themselves from method invocation delays associated with limited or absent network connectivity. Thus, applications are free to maintain many outstanding requests without blocking, and responses are provided using conventional procedure calls.

Rover provides cryptographic authentication for client-server interactions. Clients authenticate themselves and exported changes to servers; likewise, servers authenticate themselves to clients. In addition, Rover provides for safe execution of downloaded objects.

2.1 Queued RPC

Clients interact with servers using queued RPC. When a connected client fetches state from a server, the network scheduler will send the request to the server. When the reply arrives, Rover delivers a notification to the application. If the client is disconnected (or communication is too expensive) the RPC is queued until the host is reconnected; at that point all queued RPCs are sent. This asynchronous communication model decouples the application completely from the underlying communication infrastructure. During disconnected operation, the network simply appears to be very slow.

The export operation is used to propagate the changes to an object back to its owning server by sending the list of QRPCs to the server. When a QRPC is executed by the server, first a conflict detection check is performed (step 4 in Figure 1). It verifies that the object has not changed since it was imported by the application. If the object is unchanged, the modification procedure is applied to update the object (step 5a). If the object has changed, the conflict resolution procedure is invoked (step 5b). It will perform an application-specific operation, such as merging of the application's changes with the object and reporting its status to the application. Each operation executed at the server may generate information for return to the

*Authors' address: MIT Laboratory for Computer Science, 545 Technology Sq., Cambridge, MA 02139. Supervised by Professors David Gifford, M. Frans Kaashoek, and William Weihl. This work was supported in part by the Advanced Research Projects Agency under contract N00014-94-1-0985, by an NSF National Young Investigator Award, by grants from IBM and AT&T, and by an equipment grant from DEC. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government.

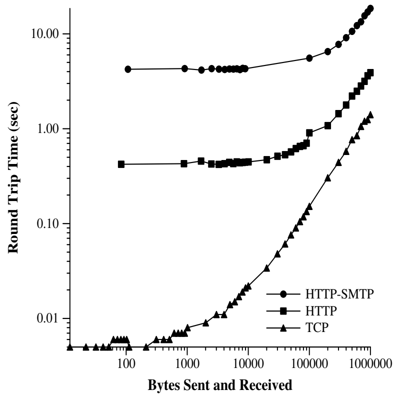

Figure 2: QRPC performance for different transport layers

application (e.g., to reflect the results of the operation, conflict resolution, or other independent changes). These results may be buffered before being returned to the application.

3 Applications

To demonstrate our ideas and the Rover toolkit, we have implemented three applications: an e-mail browser; a distributed calendar; and a Web browser.

The e-mail browser uses dynamic objects, implemented as Tcl scripts and loaded from a remote HTTP server, to create the browser interface, load the user's e-mail messages, and control application-specific consistency and prefetching strategies. While import operations are pending, the messages are marked "loading" in the client interface. The messages' states are updated as the messages arrive and are ready for viewing.

Webcal is a port of Ical, a publicly available X Window System- and filesystem-based calendar and scheduling program written by Sanjay Ghemawat. Consistency is provided on a per file (calendar) basis. In Webcal, calendars are Rover objects named by URLs. Calendars exist on servers and are cached on client hosts. Users can schedule events (i.e., modify cached calendars) while disconnected; however, the changes will be marked as tentative, and only become definite when connection is resumed. This implementation is similar to a mobile calendar tool developed for Bayou [Dem94].

The Rover Mosaic browser is the first browser that allows users to "click ahead" of the arrived data by requesting multiple new documents before earlier requests have been satisfied. The browser is an adaptation of NCSA's popular Mosaic application, with extra features for handling partially- or intermittently-connected client machines. Cached Web documents are used whenever possible, to allow for fast access in the absence of a network connection. If an uncached file is requested and the network is unavailable, an entry is created in a special locally-stored "home" page for the user, exposing the QRPC list directly to users. When a connection becomes available, the page is downloaded. Our Rover Mosaic has been used to interact with Web resources through the SMTP-based transport layer, and has been found to work reasonably well even on slow or very infrequently connected networks, as well as non-IP networks which do not permit HTTP connections.

4 Timing experiments for QRPC

To obtain insight into the performance of QRPC, we measured the underlying transport mechanisms used by QRPC. The results are presented in Figure 2. The experiments were run on two Sun workstations connected by a 10 megabit-per-second switched Ethernet carrying no other significant traffic. It was assumed that using a larger network with higher traffic would result in higher absolute latencies, but similar relative performance between different protocols.

In each test, the client sent a short request to the server, specifying the number of bytes to return. For the TCP tests, the server was a small and efficient custom C program. The server used for the HTTP tests was NCSA's HTTPD 1.3, which called a CGI program to generate the response. There are two caveats to this experiment: the NCSA server is known to be significantly (perhaps as much as 8 times) slower than some tuned commercial products; the extra process fork inccured in running a CGI script can significantly add to the execution time. We expect that a custom server for Rover would yield performance approaching that of raw TCP.

For the HTTP-SMTP tests, the HTTP client was configured to use a local proxy server running on the client machine. The proxy turned each HTTP request into an SMTP mail message and sent it to the server machine. A gateway program on the server machine intercepted the mail message, contacted the HTTP server, and mailed the result back to the client-side proxy server. The proxy then sent the HTTP response to the original client program.

The extra overhead of using SMTP gateways is obvious from Figure 2, but there are benefits which may not be so evident. One advantage is that e-mail is fundamentally a queued background process; it is more appropriate than the interactive HTTP protocol for fetching extremely large documents, such as video, which require large amounts of time regardless of the protocol. Indeed, the difference in latencies does not appear to be so pronounced as the file size grows into megabytes. Another advantage is that the IP networks required for HTTP are not always available, whereas e-mail can reach even the most obscure locations. Of course, for small documents, HTTP is still highly preferable when it is available.

The main conclusion to be drawn out of the reported experiments is that QRPC should be usable for many applications, even over electronic e-mail (SMTP).

5 Conclusion

We have found that the integration of dynamic relocatable objects and queued remote procedure calls in the Rover toolkit provides a powerful basis for building mobile applications. We have been pleased by how easy it has been to adapt applications to use these Rover facilities and thus create applications that are far less dependent on high-performance communication connectivity.

The drawback of the Rover approach is that an application architect must think carefully about how application functions can be divided between a client and a server. Queued remote procedure calls may require user interface adaptations that correspond to the idea of delayed responses to requests.

[Dem94] A. Demers *et. al.*. The Bayou architecture: Support for data sharing among mobile users. In *Workshop on Mobile Computing Systems and Applications*, 1994.

[Jos95] A. Joseph *et. al.*. Rover: A toolkit for mobile information access. Submitted for publication in SOSP '95, 1995.