Two-Level Communication Protocol for a Web Operating System (WOS™)

Gilbert Babin, Peter Kropf
Département d'informatique
Université Laval
Québec, Canada G1K 7P4
{babin,kropf}@acm.org

Herwig Unger
Fachbereich Informatik
Universität Rostock
D-18051 Rostock, Germany
hunger@informatik.uni-rostock.de

Abstract

The World-Wide Web consists not only of informational, but also computational resources. However, these resources, especially computational ones are underutilized. One characteristic of the Web is its ever changing structure; for instance, nodes are dynamically added and removed. This makes it difficult, if not impossible, to draw a complete and accurate picture of available resources. We consider the Web as a versioned system: resources, services and protocols are versioned. This paper presents a two-level protocol within this framework. The first protocol, the WOS Request Protocol (WOSRP), allows to select an appropriate version of a server. The second protocol, the WOS Protocol (WOSP), allows for locating and using these distributed (informational and computational) resources. We show how the latter protocol provides an efficient fault-tolerant resource search mechanism.

1. Introduction

With the rapid development of new forms and concepts of networked and mobile computing, it is increasingly clear that operating systems must evolve so that all machines in a given network can appear to be controlled by the same operating system. As a result, the world-wide interconnected networks, commonly called the Internet or the Web, could potentially be supported and managed by a huge virtual operating system [11]. The transparent use of heterogeneous networks of computers has been partially addressed in work on *metacomputing* [2, 3, 7], whose objectives are to transform a network into one single computer system. However, metacomputing concepts do not apply to the Web as a whole because there is no complete catalogue of all available resources and above all, such a catalogue is infeasible, given the highly dynamic and distributed nature of the Web.

To reap the potential of the Web, mechanisms are required to locate and use available, suitable resources. However, these mechanisms should consider the following requirements :

- no complete catalogue of resources is available,
- resources available are versioned (e.g., many versions of LaTeX exist),
- WOS protocols and resources may also be versioned.

In this paper, a two-level protocol is discussed to fulfill these requirements : A protocol allowing the selection of an appropriate version of WOS resources, the WOS Request Protocol (WOSRP), is detailed in Section 4. Another protocol, the WOS Protocol (WOSP), is specified in Section 5. This protocol allows to locate and use distributed resources over the Web. In the WOS, there is no distinction between 'client' and 'server'. In what follows, to simplify the definitions, we will nevertheless use:

- the term 'WOS client' when referring to a WOS node requesting resources,
- the term 'WOS server' to denote a WOS node serving a request for resources.

We also illustrate how the WOSP may be used to locate resources, such as services, machines, etc., in a fault-tolerant manner (Sect. 6). We start with a brief introduction to the concepts of a WOS in the next section followed by a discussion on the need for a two-level protocol in Section 3.

2. Concepts of a Web Operating System (WOS)

The *Web Operating System* (WOS™) [5] is a virtual operating system that supports and manages distributed/parallel processing on the Internet. The WOS is a versioned system, in which different versions not capable of dealing with a particular request for service, then pass it on to another version, as currently done for packet routing. Generalized software configuration techniques, based on a demand driven technique called *eduction* [10] are being developed, that can be used to define versions of a WOS to be built in an incremental manner. Software and hardware (description) repositories, or warehouses, will provide the necessary components for fulfilling a service requested. The kernel of a WOS is a general eductive engine responding to requests from users or other eductive engines and fulfilling these requests using its warehouses.

The WOS then works in the following manner. A request may be placed by a user to run a particular program or to initiate some service. The programs or services might be located at different sites of the network. The eductive engine may then decide whether it is capable of dealing with the request or whether it will pass it over to some other eductive engine until finally one engine accepts responsibility for the request. Once all the resources (programs, services, hardware) become available, then the program is run and the requested service fulfilled.

There are numerous projects currently under way dealing with similar problems as the WOS. Computer network exploitation approaches, such as PVM, MPI, Legion [8], CORBA [9], require login privileges and a global catalog of resources, as opposed to the WOS which uses distributed warehouses. Other approaches, including : Atlas [2], Charlotte [3], and ParaWeb [7] create a *metacomputing* infrastructure for large networks to enable parallel execution of applications. The approach proposed by WebFlow [6] exploits the Internet for parallel execution of programs.

The approaches closest to the WOS are SuperWeb [1] and WebOS [4, 13]. In SuperWeb, clients submit requests to service brokers which dispatch the requests to the appropriate servers. The model proposed includes security aspects as well as a simple economic model of resource management. Contrarily to the WOS, servers and clients must register with the broker. This implies that the broker manages a complete catalog of resources. This approach may be interesting over a not too large network, but would prove impractical over a large global network.

WebOS [4, 13] is one attempt to easily access geographically distributed resources. Specifically, it deals with the use of CPU, RAM, and disk space of resources scattered across the Internet. The approach adopted in that project is a transposition of classical operating systems problems to the reality of the Internet. The WebOS system provides solutions to the following problems :

- localization of distributed resources using SmartClient and SmartProxies,
- permanent storage using WebFS,
- remote process control,
- authentification and security using CRISIS.

The solutions proposed are highly coupled to the nodes' operating system. For instance, WebFS extends the Solaris OS. We can also point out that a global catalog of available resources is necessary. In the case of SmartClient, an applet must be downloaded from a known location to find the best available resource. In the case of SmartProxies, the client needs to explicitly know the location of the SmartProxy.

3. Rationale for a Two-Level Communication Protocol

As mentioned before, the WOS is a versioned system. In addition, no global catalogue of resources is available in the Web. Thus all methods used by the WOS must be based on the existence of decentralized warehouses as introduced in [5]. Furthermore, as described in [12], we suppose that there is a non-fixed number of such warehouses containing references to machines which potentially can execute a service in a specific area of the Web. Therefore, searching for any information implies contacting nearby warehouses. If the necessary information cannot be found there, other warehouses further away must be contacted. At least one warehouse must give a positive answer to terminate the search and initiate the service requested. This search process is applied at two levels :

- At the server level : the WOS servers are versioned. A search process should occur to identify servers supporting the same WOS versions as the client;
- At the resource level : other resources may also be versioned. Again, a search process will identify resources fulfilling the client's requirements.

A single protocol is not sufficient to handle both cases. A suitable version of the WOS server must be identified before any resource may be accessed. Remember that each client also manages a warehouse. This warehouse also contains information on available WOS servers and their versions. Once a suitable version and server is identified, a richer language is needed to request services.

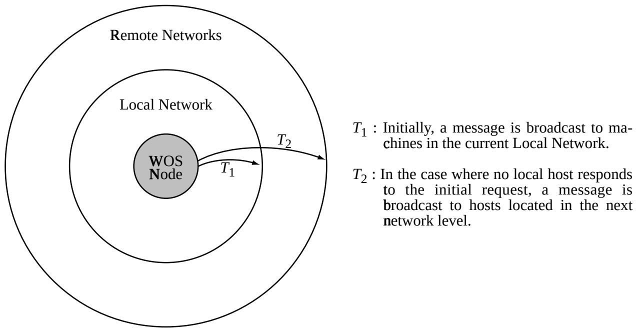
Hence, we have defined two protocols : WOSRP to identify suitable WOS servers, and WOSP to submit service requests. We present these two protocols in the following sections.

4. WOS Request Protocol (WOSRP)

WOSRP is an application-level protocol which is assumed to be used over IP networks. The rationale behind WOSRP is to provide mechanisms for WOS nodes to exchange information about WOSP versions they support. It is also used to obtain information about other WOS nodes that understand specific WOSP versions.

WOSRP has to be simple and flexible. Any version of WOSRP should be fully downwards compatible. This way, as opposed to the other approaches mentioned earlier, a machine wishing to join the network of WOS servers (WOSNet) may do so without worrying about :

- which version of WOSRP it understands;
- having to gain prior knowledge of other WOS servers;
- any administrative overhead.



T_1 : Initially, a message is broadcast to machines in the current Local Network.

T_2 : In the case where no local host responds to the initial request, a message is broadcast to hosts located in the next network level.

Figure 1. WOSRP Information Retrieval Strategy

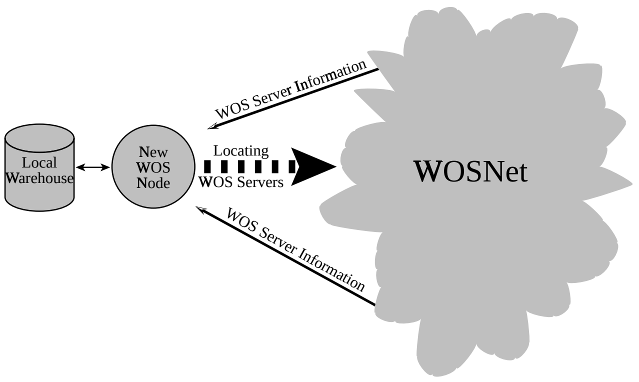


Figure 2. WOS Server Initialization Mechanism

Figures 1 and 2 illustrate how a WOS node may use WOSRP to obtain information about other WOS servers. At first, a WOS node will broadcast a request to all machines in its immediate vicinity. Any WOS server able to provide a positive answer will respond. In this case, more detailed requests may be submitted to those WOS servers. Otherwise, the WOS node broadcasts a request to all the machines at the next network level, and so on (see Fig. 1). All the responses received are used to populate the WOS node warehouse, which is at first empty when the WOS node enters WOSNet. Registration to WOSNet is implicit. The only requirement necessary is that the WOS subsystem has been previously installed (daemon, plug-in, etc.).

WOSRP uses a “pull” philosophy, where a WOS node requests information from other nodes in its vicinity. These messages may be lost without any disruption of service. Furthermore, WOS nodes may decide to propagate these messages to other sites. Eventually, replies may be returned to the node which made the original request.

These characteristics call for a datagram protocol (see Fig. 3). Requests sent do not need to be acknowledged. Eventually, replies may be received and processed by the requesting node as they arrive. The requesting node uses the replies to instantiate its warehouse.

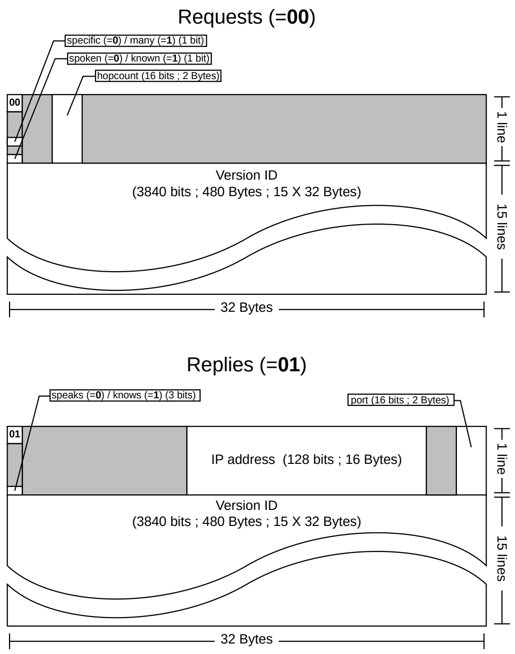

Requests (=**00**)
specific (=**0**) / many (=**1**) (1 bit)
spoken (=**0**) / known (=**1**) (1 bit)
hopcount (16 bits ; 2 Bytes)
00
1 line
Version ID
(3840 bits ; 480 Bytes ; 15 X 32 Bytes)
15 lines
32 Bytes
Replies (=**01**)
speaks (=**0**) / knows (=**1**) (3 bits)
port (16 bits ; 2 Bytes)
01
IP address (128 bits ; 16 Bytes)
1 line
Version ID
(3840 bits ; 480 Bytes ; 15 X 32 Bytes)
15 lines
32 Bytes

Figure 3. Version Requests and Replies with WOSRP

A WOS node may “speak” a certain version of the WOS protocol, which means that it can interact with other nodes using that version. A WOS node may also “know” a certain version of the WOS protocol, which means that even if it cannot interact using that version, it can refer a WOS node to other nodes which might have that capability.

WOSRP requests allow a node to question other nodes about their knowledge level (“speak” or “know”) for a specific WOS protocol version or for all the versions they speak or know.

Replies are self-contained. They indicate the knowledge level (“speak” or “know”), the IP address and port number of a WOS node which has the specified knowledge level, and a WOS Protocol version number. Each message only contains one version number. This allows for very long version numbers, which are encoded using a distributed version name registry, similar to the DNS architecture.

WOSRP also serves to establish the communication between a WOS client and a WOS server (see Fig. 4). A WOSP message may be encapsulated in a WOSRP message. This way, a generic server may receive all the requests and select the appropriate version to process them. The WOS Protocol message may be of any length. An `EOT` marker is used to indicate that the data ends. It may also be included within the data by writing it twice; in other words, the `EOT` is also the escape character.

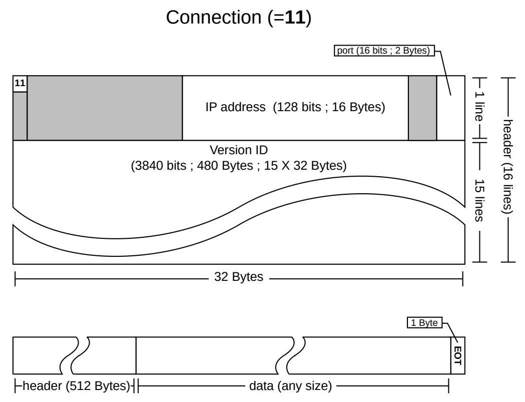


Figure 4. Connection Requests With WOSRP

5. The WOS Protocol (WOSP)

Section 3 discussed the need for a two-level Protocol approach. We have described the first level in the previous section, introducing WOSRP. We will now specify the second-level protocol, the WOS Protocol (WOSP), which is used for all interactions between WOS servers. We also describe a prototype parser for WOSP.

5.1. Specification of WOSP

WOSP allows three types commands (see Fig.5) :

1. Setup commands are used to change the execution parameters of a WOS node. For instance, a requesting node may require that the execution be completed at a certain time.

2. Execution commands allow a WOS client to use resources from another node. A WOS client, for instance, may send a command to convert some text to HTML.

3. Query commands are used by a WOS client to interrogate another WOS node's warehouse. For instance, a node might ask another node if it supports the `textToHtml` command.

```
<WOSP> ::= <requests> | <replies>

<requests> ::=  *[ <setup command> ]*
                *[ [ <execution command> | <query command> ] ]*

<setup command> ::= '@' setup_command_name 'CR-LF'
                    *[ <parameter> ]* *[ <metadata> ]*

<execution command> ::= '!' execution_command_name 'CR-LF'
                        *[ <parameter> ]* *[ <metadata> ]*

<query command> ::= '?' query_command_name 'CR-LF'
                    *[ <parameter> ]* *[ <metadata> ]*

<parameter> ::= '/*' parameter_name 'CR-LF'
                 *[ data 'CR-LF' ]* '*/' 'CR-LF'

<metadata> ::= '#*' metadata_name 'CR-LF'
               *[ data 'CR-LF' ]* '*#' 'CR-LF'

<replies> ::= 'OK' 'CR-LF'

'CR-LF' ::= carriage-return line-feed

<any>_name ::= +[ [ letters | digits | underscore | dash | period ] ]+

data ::= [ '*' [ ['\' CR-LF | ('/', '#', carriage-return, line-feed) ]
                 *[ ['\' CR-LF | (carriage-return, line-feed)] ]* |
                 '*' ['/','#']
                 *[ [ '\' carriage-return, line-feed |
                      (carriage-return, line-feed) ] ]* ]
         | [ '\' CR-LF | ('*', carriage-return, line-feed) ]
           *[ ['\' CR-LF|(carriage-return, line-feed)] ]* ]
```

Figure 5. Specification of the WOSP

Communications are connectionless. A node sends a request to another node. That node processes the commands and sends the results back to the requesting node. In the current state of WOSP, replies remain to be specified.

For each type of command, optional parameters and metadata may be transmitted. Note that the command, parameter, and metadata sets are not specified. The parameter and metadata values are encapsulated in special character sequences,`/*` and `*/`, for parameters, and `#*` and `*#`, for metadata.

In the parameter or metadata fields, lines end with the sequence carriage-return/line-feed. A new line may not begin by `*/` nor `*#`. If such is the case, an extra `*` is placed at the beginning of the line (em character stuffing). Furthermore, a \ at the end of a line is a line continuation character.

5.2. A WOSP Parser Prototype

A simple parser was developed. It implements :

- the syntax specified above;

- a set of sample services to test the capabilities of the language specified.

In favor of fast prototyping, we have relied on existing communication means to transmit messages between WOS servers; messages are encapsulated in X.400 messages, then transmitted over the Internet.

One of the services implemented embeds ASCII text into an HTML frame. A sample WOSP message requesting that service has the following pattern :

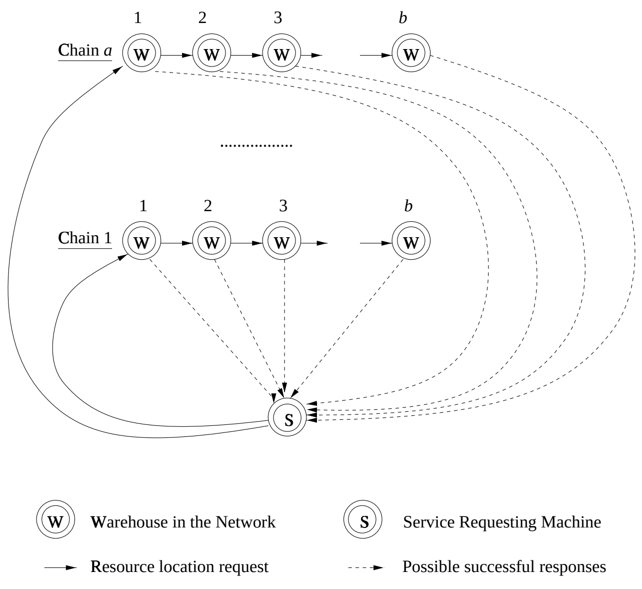
```
!textToHtml

/*text
The name of the parameter is ‘‘text’’.
A line may be continued by using the ‘‘\’’ character\
just before a CR-LF sequence.
**# is just the sequence ‘‘*#’’, while **# is the sequence ‘‘**#’’.
**/ is just the sequence ‘‘*/’’, while **/ is the sequence ‘‘**/’’.
Character stuffing only occurs at the beginning of a line !
*/
```

The `textToHtml` service takes the content of parameter `text` and embeds it in between `<body>` and `</body>` HTML commands. As a result, the service creates a file on the server's disk and sends the URL of the newly created document to the WOS client.

6. Fault-Tolerance in WOSP

In [12], we have studied approaches to efficiently locate resources in WOSNet. The approach retained searches for resources on WOS servers using multiple sequential search chains. These chains are created on demand by the WOS client, using information stored in its local warehouse. The use of these chains is illustrated in Figure 6.



1
2
3
b
Chain a
W
Chain 1
S
Warehouse in the Network
Service Requesting Machine
Resource location request
Possible successful responses

Figure 6. Sequential Search Chains

For the search to be effective, each machine in every chain must be reached, until either all the WOS servers in each chain have been visited or a WOS server able to supply the resources required has been found. This calls for a fault-tolerant approach to overcome network or server breakdowns. Only two types of failures may occur :

1. the network breaks down or times out while the message is being transmitted;

2. a WOS server cannot serve the request after receiving it.

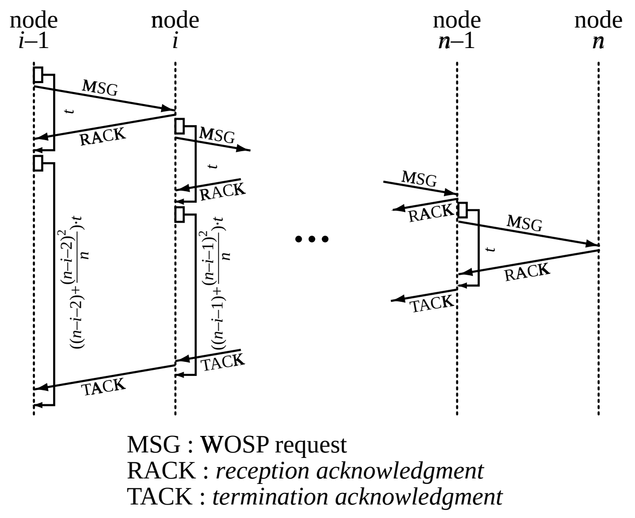


Figure 7. Locating Resources in WOSNet

To overcome these problems, we suggest the use of two types of acknowledgment messages. The first acknowledgment, the *reception acknowledgment* (RACK), confirms proper reception of the message by the next machine in the chain. The second acknowledgment, the *termination acknowledgment* (TACK), indicates that either all the WOS servers in the chain were visited or a suitable candidate was found, as shown in Figure 7.

Node i eventually receives a request from node $i-1$. The request contains (1) information about the resources being looked up, (2) the list of WOS servers in the current chain, and (3) the address of the WOS client initiating the request. Node i first sends a RACK before a time-out of length t expires on node $i-1$. Node i then checks whether it can provide the requested resources. In the affirmative, it replies to the WOS client and sends a TACK to node $i-1$. Otherwise, the search continues in the same manner to the next node in the list. Assuming the chain contains n WOS servers, the TACK must reach node $i-1$ before a time-out of length $((n-i-2)+\frac{(n-i-2)^2}{n})\cdot t$ expires. At node $n-1$, the length of that time-out is 0; in this case, the WOS server assumes a TACK on receiving the RACK.

If any time-out occurs at node i, node i terminates the chain from nodes $i+1$ to n and restarts it with node $i+2$. If the time-out occurs at node $n-1$, the search in that chain is simply terminated.

Many factors may play a role for deciding whether or not a node can supply the requested resources:

- strategies and policies for sharing the node's resources over WOSNet;

- current workload;

- WOS clients requirements.

7. Future Work

So far, we have set up an experimental environment to test the WOS concepts and solutions. In order to bring the WOS to an operational level, work on following tasks has been initiated :

- Transformation of communication links from X.400 messages to TCP/IP system calls;
- Separation of the WOSP parser from the WOS services;
- Creation of the eductive engine;
- Implementation of a larger number of services;
- Specification of the WOSP version space;
- Development of version management approaches and techniques.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC grants #OGP0155899 and #OGP0170164).

References

[1] A. Alexandrov, M. Ibel, K. Schauser, and C. Scheimann. Superweb: Research issues in java-based global computing. In *Workshop on Java for Computational Science and Engineering Workshop*, Syracuse University, Dec. 1996.

[2] J. Baldeschwieler, R. Blumofe, and E. Brewer. Atlas: An infrastructure for global computing. In *Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

[3] A. Baratloo, M. Karaul, Z. Kedem, and P. Wykoff. Charlotte: Metacomputing on the web. In *9th Conference on Parallel and Distributed Systems*, 1996.

[4] E. Belani, A. Vahdat, T. Anderson, and M. Dahlin. The CRISIS wide area security architecture. In *Seventh USENIX Security Symposium (Security 98)*, San Antonio, TX, USA, Jan. 1998.

[5] S. Ben Lamine, P. Kropf, and J. Plaice. Problems of Computing on the Web. In A. Tentner, editor, *High Performance Computing Symposium 97*, pages 296 – 301, Atlanta, GA, April 1997. The Society of Computer Simulation International.

[6] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran. Webflow. In *Workshop on Java for Computational Science and Engineering Workshop*, Syracuse University, Dec. 1996.

[7] T. Brecht, H. Sandhu, M. Shan, and J. Talbot. Towards world-wide supercomputing. In *Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

[8] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds. A synopsis of the Legion project. Technical Report CS-94-20, University of Virginia, June 1994.

[9] T. Mowbray and R. Zahavi. *The Essential CORBA: Systems Integration Using Distributed Objects*. John Wiley & Sons, New York, NY, USA, 1995.

[10] J. Plaice and S. Ben Lamine. Eduction: A general model for computing. In *Intensional Programming II*. World Scientific, Singapore, 1997.

[11] F. Reynolds. Evolving an operating system for the Web. *IEEE Computer*, 29(9):90–92, 1996.

[12] H. Unger, P. Kropf, G. Babin, and T. Böhme. Simulation of search and distribution methods for jobs in a Web operating system (WOS^{TM}). In *1998 Advanced Simulation Technologies Conference (ASTC 1998)*, Boston, Massachusetts, USA, Apr. 1998.

[13] A. Vahdat, E. Belani, P. Eastham, C. Yoshikawa, T. Anderson, D. Culler, and M. Dahlin. WebOS: Operating system services for wide area applications. Technical Report CSD-97-938, University of California, Berkeley, Dec. 1997.