

Short Paper

WSNFS: A Web-Services-Based Network File System *

GWAN-HWAN HWANG¹, CHIH-HAO YU², CHUN-CHIN SY³ AND CHIU-YANG CHANG³

¹*Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, 106 Taiwan*

²*Networks and Multimedia Institute
Institute for Information Industry
Taipei, 106 Taiwan*

³*Department of Electronic Engineering
National United University
Miaoli, 360 Taiwan*

Web services are Web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data between calling clients and servers. In this paper, we demonstrate that Web services can be used to construct a distributed file system, called the Web-services-based network file system (WSNFS). One of the goals of the WSNFS is to provide a platform for file sharing among heterogeneous distributed file systems. Also, as the technology of Web services is widely applied to the field of grid computing, the proposed system can aid the deployment of Web services to the data grid applications. We present the architecture of the WSNFS and define the communication-protocol standard based on Web services. Our experiences in implementing the prototype as well as the experimental results demonstrate that the practicality of the WSNFS; the performance of the prototype is comparable to that of the Sun network file system.

Keywords: distributed file system, network file system, web services, XML, SOAP

1. INTRODUCTION

Distributed file systems support the sharing of information in the form of files throughout an intranet. A well-designed distributed file system provides access to files stored on a server with performance and reliability similar to files stored on local disks. A distributed file system enables programs to store and access remote files exactly as they do local ones, allowing users to transparently access files from any computer in an intranet [1]. Many types of distributed file system have been widely used for more than a decade, including the Sun network file system (NFS) [2], the Andrew file system (AFS) [3], the distributed computing environment [4], the UFO global file system [5], the Coda

Received October 20, 2006; revised January 17 & July 18, 2007; accepted August 3, 2007.

Communicated by Tei-Wei Kuo.

* The preliminary result of this research was presented in International Conference on Internet and Web Applications and Services (ICIW), February 23-25, 2006, Guadeloupe, French Caribbean. This research was supported in part by the National Science Council of Taiwan, R.O.C. under grants No. 94-2213-E-003-006 and 95-2221-E-003-007.

file system [6], and the Samba file system [7]. WebDAV (Web-based Distributed Authoring and Versioning, also called DAV) is a set of extensions to HTTP/1.1 allowing the user to access documents on a remote Web server. It provides support for editing, setting the properties, and locking of the documents [8]. It provides some functionalities which are also available in the network file systems. One of the major differences between WebDAV and ordinary network file systems is that the legacy application programs should be modified to access the remote files in the WebDAV. The application programs cannot access the files as they are local files.

Technological advances in device miniaturization and wireless networking have led to the increasing integration of small and portable devices into distributed systems. These devices include laptop PCs, handheld devices, wearable devices, and devices embedded in appliances. Many such devices have low hardware resources (*e.g.*, in terms of disk space), but must be supported given that users are becoming accustomed to the benefits of the resource sharing provided by distributed file systems. Fig. 1 (a) shows a scenario in which the user can connect to heterogeneous distributed file servers. Several difficulties arise when deploying this type of architecture. First, the client has to install the driver module and management tools for each type of distributed file system, such as the individual drivers in the kernel of its operating system for the NFS and AFS. The binary codes of these driver modules have to be stored in the local disk of the client, and they all must reside in the memory space of the client device even when the client is not connected to all of the file servers. Second, the client has to install a new driver module whenever connecting to a new type of file server. Third, porting these driver modules to a new client device usually involves considerable effort since they usually have to be installed in the system kernel of the client. Fourth, some distributed file systems restrict the types of mobile client devices that can connect to them. For example, the client must have a fixed IP address to connect to an NFS file server. Also, the presence of a firewall may interfere with the operation of some types of network file systems.

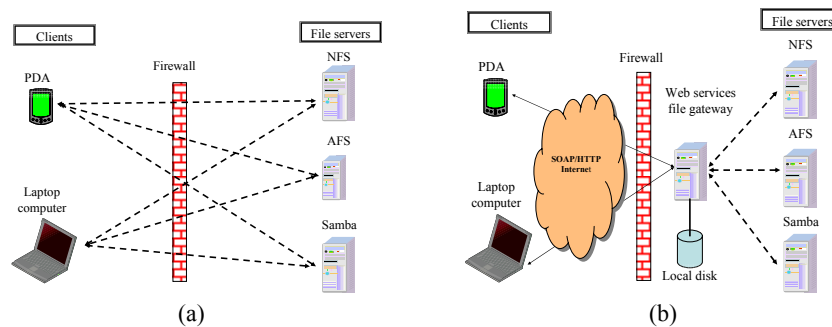


Fig. 1. Resource sharing by connecting to several types of file server.

Fig. 1 (b) shows the network file system proposed in this paper, which is called the Web-services-based network file system (WSNFS). A Web services file gateway is installed to overcome the problems associated with file sharing among heterogeneous file servers. All of the file directories of the file servers that are to be shared by the client are mounted on the client. The client sends file-operation requests to the gateway, which is

responsible for forwarding each request to the appropriate file server and returning the result to the client. The communication between the client and the gateway follows the standard of XML Web services [9].

XML Web services is a new technology that promises greater ease-of-use and interoperability than previous distributed computing technologies such as DCOM [10], CORBA [11], and Java RMI [12], through the use of industry-standard XML protocols such as SOAP [13], Web service description language (WSDL) [9], and UDDI [9]. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used to describe the services available, and UDDI is used to list the available services. One of the advantages of employing Web services to represent the request and response messages between clients and servers is that the Web services allow different applications from different sources to communicate with each other without requiring time-consuming custom coding. Also, because all communications are performed in XML, the Web services are not tied to any particular operating system or programming language. For example, Java can communicate with Perl, and MS Windows applications can communicate with UNIX applications.

Furthermore, as the software toolkits used for building grid-computing environments make considerable use of Web services technology [14], our system can be easily applied to the grid computing. There are currently two major file-sharing mechanisms in the grid-computing field. The first employs traditional distributed file systems such as the NFS and AFS. In this case, the system faces the problems we mention above. The other mechanism is assisted by the software package gridFTP [15], the main drawback of which is that the client must download an entire file before accessing any part of it, whereas in some situations only a small part of the target file is needed.

With the architecture shown in Fig. 1 (b), the drawbacks of the system shown in Fig. 1 (a) disappear. First, only a single driver module needs to be installed in the client device, which reduces its disk and memory requirements. Second, incorporating a new type of file server into the system only requires the installation of the new driver module of the new file server in the WSNFS file gateway. Third, the above two advantages are due to only the driver module for communication with the gateway being installed in the client device, which reduces the work in porting the driver module in the client. Finally, the Web services adopt the SOAP that uses HTTP as the transport mechanism, and since most firewalls allow the passage of HTTP traffic our network file system can work well without needing to reconfigure the firewall.

The remainder of the paper is organized as follows. Section 2 discusses the system architecture and communication protocol of the WSNFS, section 3 presents the details of our implementation of it, section 4 describes the experimental results, and section 5 presents the conclusions that can be drawn from this paper.

2. SYSTEM ARCHITECTURE AND COMMUNICATION PROTOCOL

In this section we describe the system architecture of the WSNFS. Our goal is to define a standard communication protocol for the WSNFS to which the client and gateway conform, thus making them machine independent.

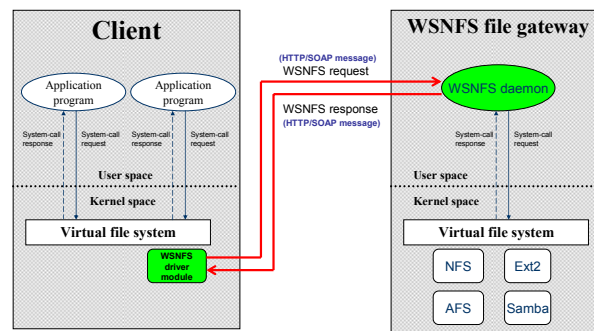


Fig. 2. Architecture of the WSNFS.

2.1 System Architecture

Fig. 2 shows the architecture of the WSNFS. First, the file directories that the client wants to share are mounted on the WSNFS driver module installed in the system kernel of the client device. Second, when the application program executed in the client device invokes a system call related to file operations, the file system checks if this call requires access to the directories mounted on the WSNFS driver module. If such access is required, the file system sends the parameters of the invoked system call to the WSNFS driver module. Third, the WSNFS driver module translates the invoked system call to conform to the standard of a POSIX.1 (ISO/IEC 9945-1:1990, IEEE Std.1003.1) I/O system-call interface [16]. Fourth, the POSIX.1-formatted system call is packed into a SOAP message based on the standard of Web services. This produces the WSNFS request shown in Fig. 2. Note that the file servers are also mounted on the gateway, as shown in Fig. 1 (b). Fifth, the WSNFS daemon of the gateway receives the request and retrieves the name and parameters of the invoked system call, and then makes the appropriate system call. Sixth, the system call returned from the gateway file system is packed into a SOAP message and sent to the WSNFS driver module of the client. Finally, the module unpacks the message and the file system of the client sends a system-call response to the application program that made the system-call request.

In Fig. 2, all the application programs need not to do any modification to access the file systems in the WSNFS file gateway, as the system calls for file accesses will be re-directed to the WSNFS driver module. For a client without the supporting of WSNFS driver module in its operating system, its application programs should invoke the WSMFS requests and receive WSNFS responses directly to perform file operations in the WSNFS file gateway. It can be supported by a program library.

2.2 Communication Protocol Between the Client and WSNFS Gateway

The contents of the WSNFS request and response according to the POSIX.1 I/O system-call interface are listed in Table 1. However, because a SOAP message is an XML-formatted data, the presence of many items may result in the SOAP message containing many tag delimiters, which makes the message large and slow to parse. Moreover, some of the output parameters of WSNFS responses have a complicated data structure. In

Table 1. Commands and parameters in WSNFS requests and responses.

Command of WSNFS request	Input parameters of WSNFS request	Output parameters of WSNFS response
lstat	path	res, error, statbuf (Refer to [18])
statfs	path	bufsize, res, error statfsbuf (Refer to [18])
opendir	path	dpstat, res, dp
readdir	dp, offset	item ((Refer to [18])
closedir	dp	res, erro
readlink	path, size	error, res, readlinkbuf
mknod	path, mode, redv	res, erro
mkdir	path, mode	res, erro
unlink, rmdir	path	res, erro
rename, symlink	from, to	res, erro
chown	path, uid, gid	res, erro
chmod	path, mode	res, erro
utime	path, buf	res, erro
truncate	path size	res, erro
close, fsync	df	res, erro
open	path, flags	fd, erro
read	fd, size, offset	res, erro, buf (base64 encoded)
write	fd, buf (base64 encoded), size, offset	res, erro

this case, to reduce the number of items in the SOAP message of the WSNFS response, they are packaged into binary format and encoded to base64 code [17]. For the detailed data structure of these records, refer to [18]. Since the WSNFS request and response conform to the standard of Web services, we define their format using the WSDL. A complete definition of WSNFS requests and responses in WSDL can be downloaded from the Internet [18].

Below we give examples of WSNFS requests and responses. The standard of the SOAP message does not provide an authentication mechanism, and hence in our current design we employ the HTTP authentication available in HTTP/1.0 [19]. This depends on checking the username and password string that are embedded in the HTTP headers when a client makes an HTTP request. The username and password strings are encoded in base64. Fig. 3 shows an example of the “open” WSNFS system-call request. The seventh line of the code is the base64-encoded username and password strings. The SOAP message appears after line 10. Figs. 3 and 4 are examples of WSNFS requests and responses for “open” system calls, respectively. For more examples such as “read” and “write” system calls, refer to [18].

According to the POSIX.1, when a system call discovers an error, it returns - 1 and stores the reason the called failed in an external variable named “errno.” In Unix system, the “/usr/include/errno.h” file maps these error numbers to manifest constants. Since, the system-call requests issued by the application programs are delivered in HTTP protocol, the execution of these system calls introduce additional errors which are related to the HTTP protocol. The POSIX.1 also defines operational errors for the IPC/network software. The WSNFS complies with the error numbers defined in POSIX.1. For example, the error of connection time out in the WSNFS request is the ETIMEDOUT defined in POSIX.1.

```

POST / HTTP/1.1
Host: 140.122.76.177:8080
User-Agent: gSOAP/2.7
Content-Type: text/xml; charset=utf-8
Content-Length: 458
Connection: keep-alive
Authorization: Basic aG93aWU6dmlzaXQ=
SOAPAction: ""
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ws="http://tempuri.org/ws.xsd">
<SOAP-ENV:Body>
  <ws:open>
    <path>/home/howie/document/doc.txt</path>
    <flags>32768</flags>
  </ws:open>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Fig. 3. Example of an “open” WSNFS system-call request.

```

HTTP/1.1 200 OK
Server: gSOAP/2.7
Content-Type: text/xml; charset=utf-8
Content-Length: 423
Connection: keep-alive
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ws="http://tempuri.org/ws.xsd">
<SOAP-ENV:Body>
  <ws:openResponse>
    <error>0</error>
    <fd>9</fd>
  </ws:openResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Fig. 4. Example of an “open” WSNFS system-call response.

3. IMPLEMENTATION DETAILS OF WSNFS

In this section we present our implementation of the WSNFS to demonstrate its feasibility. We describe how to implement the WSNFS daemon and the WSNFS driver module in sections 3.1 and 3.2, respectively. Note that both the WSNFS gateway and the client device execute the Linux operating system [20].

3.1 Implementation of the WSNFS Daemon

We employ the gSOAP, which is a Web service development toolkit [21]. The gSOAP offers an XML-to-C/C++ language binding to ease the development of SOAP/XML Web services in C and C/C++. First, we write the header file of the service-interface description in C/C++ according to the communication protocol defined in section 2 for gSOAP [21]. We then use the gSOAP stub and skeleton compiler to generate the gSOAP stub and gSOAP skeleton according to the header file. Note that only the gSOAP skeleton is used in the WSNFS daemon. A system-call invoker is also implemented. The gSOAP skeleton calls the corresponding method in the system-call invoker when it receives the SOAP request from the client device. Details on the header file and the system-call invoker are available elsewhere [18].

3.2 Implementation of the WSNFS Driver Module

The FUSE provides a simple interface for user-space programs to export a virtual file system to the Linux kernel [22]. It also aims to provide a secure method for users without system privileges to create and mount their own file systems. The FUSE driver is embedded into the kernel of the client device, and will invoke functions of a user-space program called the “FUSE user-level implementation” (FULI). The FULI is designed to invoke functions in the gSOAP stub obtained as described in section 3.1 according to the communication protocol defined in section 2. Details on the implementation of the FULI are available elsewhere [18].

We have developed a scheme for reducing the overhead of message transmission in the network. This is because some application programs tend to issue consecutive write operations involving small amounts of data. We therefore set up write buffers in the FULI. As there are several consecutive write operations to the same file (associated with the same file descriptor), we combine them into a single write operation. For each write operation, the FULI stores the received operation and immediately sends a response to the FUSE driver. We call this scheme a delayed-write buffer. A 64KB buffer in the FULI is allocated to each open operation for a file. The FULI has to flush the buffer when the buffer is full, when the application program issues a release (close) operation for this file, and when another application program issues an open operation for this file.

4. EXPERIMENTAL RESULTS

In this section we present the results of the performance evaluation of the prototype of the WSNFS we implemented. Applying the SOAP to represent the messages resulted in some overheads. First, since a SOAP message is also an XML document, the software system needs to parse the document to retrieve the information therein. Second, an XML document is usually 4-10 times the size of the original binary data it represents due to the presence of additional tag delimiters and the encoding of the binary data in base64 format [23].

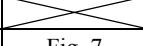
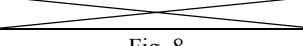

Many studies have evaluated the performance of Web services in diverse fields, such as scientific computing and grid computing [23-25]. In order to determine whether

the WSNFS can meet the performance requirements of a distributed file system, we compared its performance with that of a widely used distributed file system, Sun NFSv3 [2]. We employed a file system benchmark tool called Bonnie [26] to evaluate the performance of the following configurations (see Table 2): there are abbreviated to “local”, “nfs”, “basic”, “cache”, and “ca + lr + dw”. The “cache” configuration involves activating the kernel cache function of the FUSE driver. In addition to the kernel cache, in the “ca + lr + dw” configuration we further activate the large-read function in FUSE [27] and the delayed-write-buffer function.

Table 2. Configurations of the evaluated systems.

Configuration	Description
Local file system (<i>local</i>)	File operations are performed on the local disk.
NFS (<i>nfs</i>)	An NFS server running the Linux fedora core 1 with the 2.4.22-1.2199.nptlsmp kernel.
WSNFS (<i>basic</i>)	The prototype WSNFS that we implemented as described in section 2.
WSNFS with kernel cache (<i>cache</i>)	The prototype WSNFS with kernel caching in the setting of the FUSE driver [22].
WSNFS with kernel cache and delayed-write buffer (<i>ca + lr + dw</i>)	The prototype WSNFS with kernel caching, large-read buffer of FUSE, and delayed-write buffer.

Table 3. Network environments and types of client devices in the experiments.

Client types Network environments	Client A: Desktop PC	Client B: Laptop PC	Client C: Laptop PC with wireless network card
Local host	Fig. 5		
LAN	Fig. 6	Fig. 7	Fig. 8
WAN	Fig. 9	Fig. 10	

We measure the performance of the configurations in Table 3 in three different network environments with a variety of client devices (see Table 3). In the “local host” environment, the WSNFS and the client device are the same machine. In the “LAN” environment, the client device and the WSNFS gateway are in the same network segment and connected by a 100-Mbps network switch. In the “WAN” environment, network packets are transmitted between the client device and WSNFS through five network routers. The server machine is a PC with a 3.2-GHz Pentium-4 CPU, 1 GB of RAM, and 512 KB of level 2 cache, and running Fedora core 1 with the 2.4.22-1.2199.nptlsmp kernel. The mount point on the server is on an 80-GB 7200-rpm IDE disk formatted with the Ext3 file system.

Three types of client devices were tested: client A is the same as the WSNFS gateway mentioned previously, whereas clients B and C are laptop PCs with a 650-MHz Pentium 3 CPU and 128 MB of RAM, and running Linux fedora core 1. Clients B and C are equipped with a 100-Mbps Ethernet card and a wireless 802.11b PCMCIA card, respectively. Figs. 5 to 10 show the measured performance results. The access files had sizes of 1,

Table 4. Performance of six file operations as measured by Bonnie.

I/O type	I/O unit	Descriptor
Sequential output	Per character (<i>Write char</i>)	The file is written using the <i>putc()</i> stdio macro.
	Block (<i>Write block</i>)	The file is created using <i>write(2)</i> .
	Rewrite (<i>Rewrite</i>)	Each chunk (of size 16384) of the file is read with <i>read(2)</i> , dirtied, and rewritten with <i>write(2)</i> , requiring an <i>lseek(2)</i> .
Sequential input	Per character (<i>Read char</i>)	The file is read using the <i>getc()</i> stdio macro. Once again, the inner loop is small. This should require only standard I/O functions and sequential input.
	Block (<i>Read block</i>)	The file is read using <i>read(2)</i> . This should be a very pure test of sequential input performance.
Random seek	(<i>Seek</i>)	This test runs four processes in parallel, performing a total of 4000 <i>lseek()</i> commands to various locations in the file computed. In each case, the block is read with <i>read(2)</i> . In 10% of cases, it is dirtied and written back with <i>write(2)</i> .

10, 50, and 100 MB. We employed the Bonnie benchmark tool to measure the six types of I/O operations notated as “Write char”, “Write block”, “Rewrite”, “Read char”, “Read block”, and “Seek” in Figs. 5 to 10. The details of these operations are given in Table 4.

In Fig. 5, the file operations are performed on local host without the network transmission. It goes without saying that the local file system is with the best performance. In “Write char”, the NFS and WSNFS can reach about 33% and 23% performance of the local file system, respectively. In “Write block”, the NFS and WSNFS can reach about 10% and 5% performance of the local file system, respectively. The tremendous decrease of performance in the write operation may be due to the reason that the disk cache of the local file system is supported by the operating system directly. However, in the read operations, “Read char” and “Read block”, the performance of NFS is very closed to the local file system. The WSNFS can obtain about 76% performance of local file system. Generally speaking, the NFS and WSNFS have better performance in both read and write related operations for large files. It is because the network file system needs to wrap the commands and data in network packages. For file operations to small files, the overhead becomes larger. In the “Seek” operation, the NFS and WSNFS can only reach about 2.6% and 6.8% performance of the local file system, respectively. Since the execution of “Seek” of WSNFS is only better than that of NFS when the kernel cache is enabled, we may consider that the use of cache is critical for the “Seek” operation. For the three different settings of WSNFS, “basic”, “cache”, and “ca + lr + dw”, the first and last one always have the worst and best performance, respectively.

In Fig. 6, all the file operations make network communications via a 100-Mbps network switch. Compared to the result in Fig. 5, the performance degrades significantly. In the write operations, the WSNFS can obtain about 35% to 50% performance of the NFS for files whose sizes are less than 50MB. However, for large files, 100MB, the WSNFS has 70% performance of the NFS. For the read operations, the WSNFS has almost the same performance of NFS in case its kernel cache is enabled. Also, the WSNFS seems to have better performance in “Seek” when its kernel cache is enabled. Figs. 5 and 6 clearly indicate that the kernel cache and delayed-write buffer of FUSE can boost the

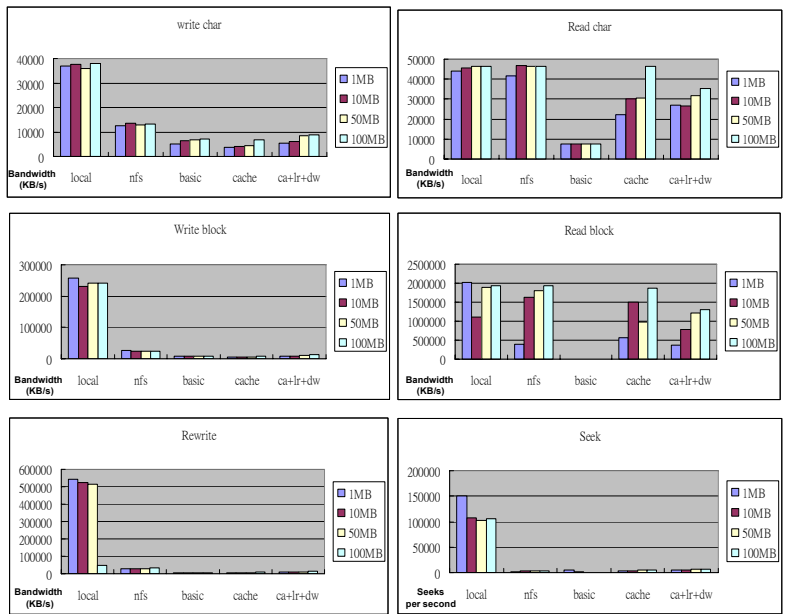


Fig. 5. Experimental results: local host and client A.

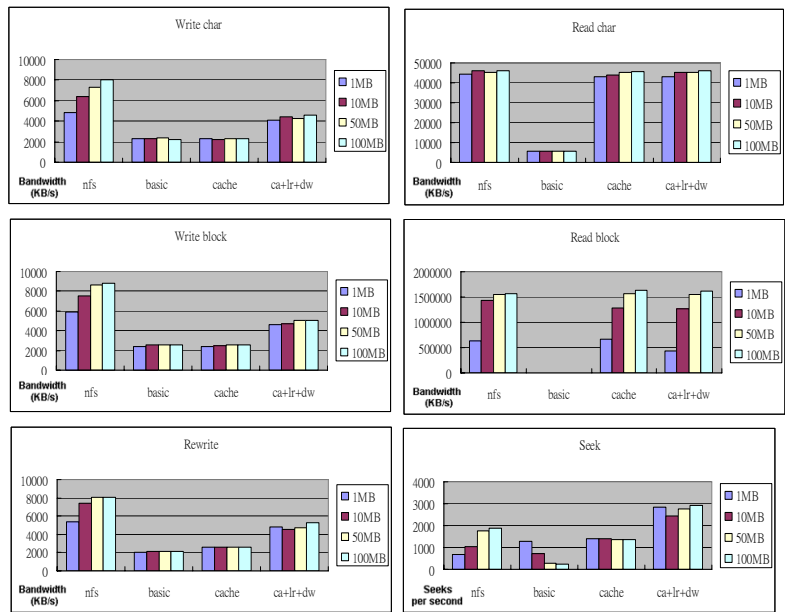


Fig. 6. Experimental results: LAN and client A.

performance of the WSNFS significantly. Since the NFS does not have the overhead of resolving the SOAP message, its performance is better than the WSNFS. Note that one of the major overheads in resolving the SOAP message is the base64 decoding and encoding.

In Fig. 7, we conduct similar experiments as we have done in Fig. 6. However, the client device is with less computing power. We can find out that the performance of write operations of WSNF is degraded compared to NFS. It is because the overhead for SOAP message resolution is higher for a client with less computing power. The degradation of read operations are not significant compared to the write operations. In Fig. 8, we still conduct experiments for client device which is with less computing power. The communication between file server and the client device is via 802.11b wireless protocol. Compared to the result in Fig. 7, the write operations of NFS and WSNFS are about 50 % performance; the read operations of NFS and WSNFS are about 25% performance. Also, we find out that the performance of read operations of both NFS and WSNFS degrade tremendously when the files are larger then 100 MB. In [28], researchers conduct experiments with NFS over wireless link. They found out that it displayed a dramatic decrease in performance when burst errors were introduced. Their results seem to conform to that shown in Fig. 8.

Figs. 9 and 10 show that the WSNFS still exhibits acceptable performance when operating across an intranet. As the performance of network in the wide area environment could be influenced by a lot of factors, we find some situations which is difficult to explain. For example, the performance of WSNFS is better than the NFS in some cases. What we realize from the two figures is that the delay write buffer seems to improve the performance. It is obvious as it can reduce the amount of the network transmission.

Figs. 5 and 6 clearly indicate that the kernel cache and delayed-write buffer of FUSE can boost the performance of the WSNFS significantly. The overhead for SOAP message resolution is higher for a client with less computing power, as shown in Figs. 7 and 8. However, the WSNFS can still reach about 50% of the performance of the NFS in general case. Figs. 9 and 10 show that the WSNFS still exhibits acceptable performance when operating across an intranet.

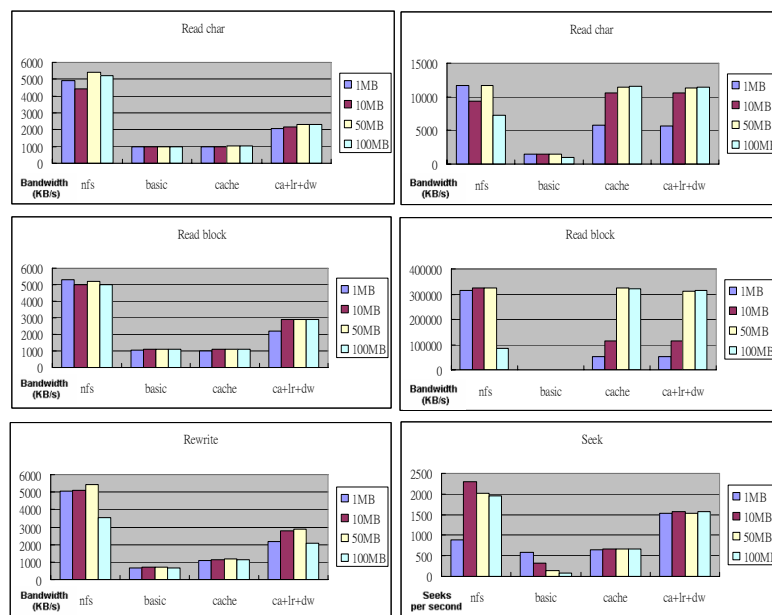


Fig. 7. Experimental results: LAN and client B.

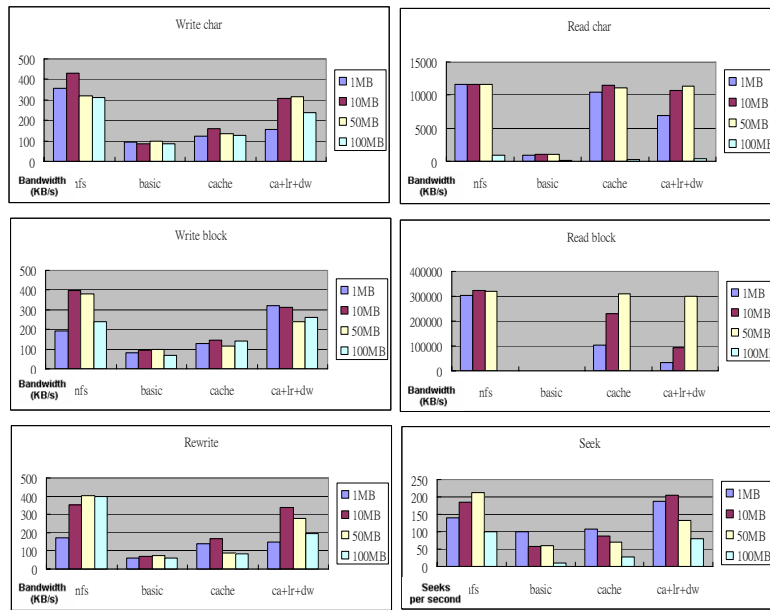


Fig. 8. Experimental results: LAN and client C.

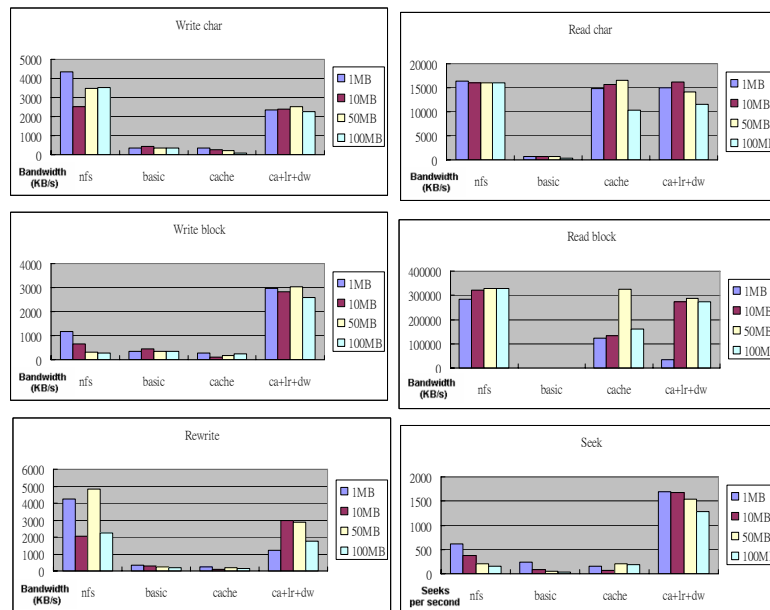


Fig. 9. Experimental results: WAN and client A.

Another performance experiment involved playing a VCD 2.0 movie file with the multimedia player application “mplayer.” The network work environment was set to the “WAN” (see Table 3). The server machine is a PC which is the same as the file server used in conduct experiments from Fig. 5 to Fig. 10. The client device is the “Client A” as

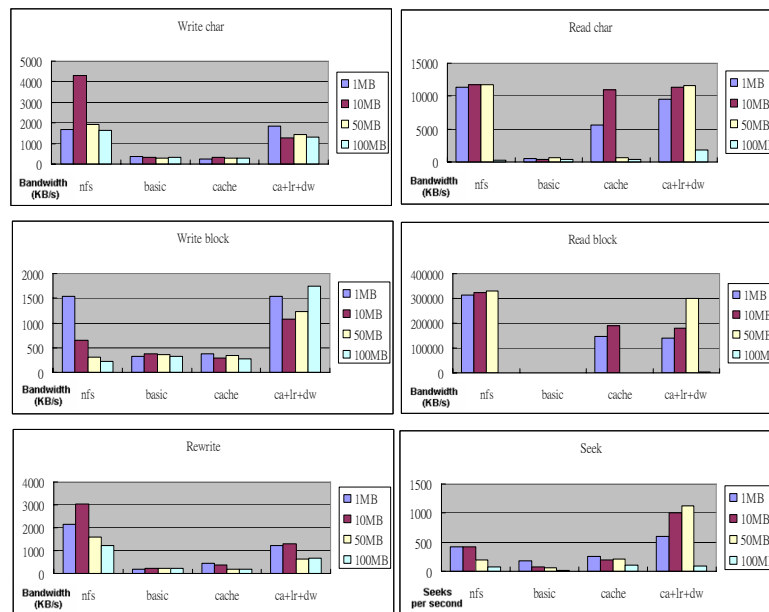


Fig. 10. Experimental results: WAN and client B.

shown in Table 3. We connected ten client devices to the file server simultaneously. This multimedia application could play the movie file mounted by the WSNFS at 25 frames/second without dropping any frames. This demonstrates the practical feasibility of the WSNFS.

5. CONCLUSIONS AND FUTURE WORK

We consider that the system presented in this paper provides a platform for solving the problems associated with the use of mobile devices to share files in a heterogeneous distributed system. The system can be applied to grid computing, since the Web services it uses are becoming a standard for remote-procedure calls. The reported experimental results demonstrate the feasibility of the proposed system in a variety of network environments.

The following work must be performed. First, we only implemented the proposed system in Linux machines, whereas the WSNFS has cross-platform interoperability. The WSNFS client should therefore be ported to platforms such as Windows XP, Windows CE, and Palm OS. Second, the authentication model in the current design relies on HTTP authentication available in HTTP/1.0, whereas each WSNFS request should be authenticated by the WSNFS file gateway. This would increase the authentication overhead, but one solution is to provide session-based authentication in which only a single authentication is made for file system calls between paired open and close system calls. Also, implementing secure transfers has not been discussed here; encryption by cryptographic algorithms also needs to be implemented. Third, we have not addressed some of the other issues that are important in the majority of distributed systems, such as server crash re-

covery, ensuring data consistency, and fault tolerance. Finally, several methods to reduce the running time of the transaction of Web services were proposed. In [29], it addresses how to improve Web services' performance by compressing SOAP. Also, a variety of methods to improve the performance of Web services are discussed in [30]. We do not employ these schemes to improve the performance of WSNFS.

REFERENCES

1. G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: Concepts and Design*, 3rd ed., Addison-Wesley, 2001.
2. B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS version 3: design and implementation," in *Proceedings of the Summer USENIX Technical Conference*, 1994, pp. 137-151.
3. J. H. Howard, "An overview of the Andrew file system," in *Proceedings of the USENIX Winter Technical Conference*, 1998, pp. 23-26.
4. M. L. Kazar, B. W. Leverett, O. T. Anderson, V. Apostolides, B. A. Bottos, S. Chutani, C. F. Everhart, W. A. Mason, S. T. Tu, and E. R. Zayas, "Decorum file system architectural overview," in *Proceedings of the Summer USENIX Technical Conference*, 1990, pp. 247-259.
5. A. D. Alexandrov, M. Ibel, K. E. Schauer, and C. J. Scheiman, "Extending the operating system at the user level: the UFO global file system," in *Proceedings of the USENIX Annual Technical Conference*, 1997, pp. 77-90.
6. L. B. Mummert, M. R. Ebling, and M. Satyanarayanan, "Exploiting weak connectivity for mobile file access," in *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995, pp. 143-155.
7. J. R. Vernooij, J. H. Terpstra, and G. J. Carter, "The official samba-3 HOWTO and reference guide," <http://us3.samba.org/samba/docs/man/Samba-HOWTO-Collection/>.
8. R. Dornfest, "Emerging technology briefs: WebDAV," <http://webservices.xml.com/pub/a/ws/2002/03/26/webdav.html>.
9. World Wide Web consortium, "Web services activity," <http://www.w3.org/2002/ws/>.
10. E. Frank and III Redmond, *DCOM: Microsoft Distributed Component Object Model*, Hungry Minds Inc., 1997.
11. OMG, "Common object request broker architecture," <http://www.omg.org>, 1995.
12. "Java remote method invocation (Java RMI)," <http://java.sun.com/products/jdk/rmi/>.
13. D. Box *et al.*, "Simple object access protocol (SOAP) 1.1. W3C Note 08 May 2000," <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
14. "The globus toolkit homepage," <http://www.globus.org/toolkit/>.
15. GridFTP, <http://www.globus.org/datagrid/gridftp.html>.
16. 2004 IEEE Standard for Information Technology – Portable Operating System Interface (POSIX®) – Technical Corrigendum 2, ISBN: 0-7381-3987-4, 2004.
17. N. Freed and N. Borenstein, "Multipurpose internet mail extensions (MIME) part one: format of internet message bodies," <http://www.ietf.org/rfc/rfc2045.txt>, 1996.
18. C. H. Yu, "Web services based network file system," Master Thesis, Dept. of Information and Computer Education, National Taiwan Normal University, Taiwan, 2005.
19. J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1," <http://www.ietf.org/rfc/rfc2616.txt>, 1999.
20. Linux Online Inc., "Linux online," <http://www.linux.org/>.

21. "gSOAP: C/C++ web services and clients," <http://gsoap2.sourceforge.net/>.
22. "Filesystem in userspace," <http://fuse.sourceforge.net/>.
23. F. E. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, "Efficient wire formats for high performance computing," in *Proceedings of the Conference on Supercomputing*, 2000, pp. 139-147.
24. K. Chiu, M. Govindaraj, and R. Bramley, "Investigating the limits of SOAP performance for scientific computing," in *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, 2002, pp. 56-61.
25. D. Davis and M. Parashar, "Latency performance of SOAP implementations," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002, pp. 407-412.
26. "Bonnie – File system benchmarks," <http://cc.jlab.org/docs/scicomp/benchmark/bonnie.html>.
27. README of FUSE, <http://cvs.sourceforge.net/viewcvs.py/fuse/fuse/README?rev=1.11>.
28. R. Dube, C. D. Rais, and S. K. Tripathi, "Improving NFS performance over wireless links," *IEEE Transactions on Computers*, Vol. 46, 1997, pp. 290-298.
29. M. Nikitas, "Improve XML web services' performance by compressing SOAP," <http://www.dotnetjunkies.com/Article/46630AE2-1C79-4D5F-827E-6C2857FF1D23.dcik>.
30. "Java performance tuning," <http://www.javaperformancetuning.com/tips/webservice.shtml>.

Gwan-Hwan Hwang (黃冠寰) is an associate professor in the Department of Computer Science and Information Engineering at National Taiwan Normal University, Taiwan. He received the B.S. and M.S. degrees while in the Department of Computer Science and Information Engineering at National Chiao Tung University, in 1991 and 1993, respectively, and the Ph.D. degree while in the Department of Computer Science at National Tsing Hua University, Hsinchu, Taiwan, in 1998.

Chih-Hao Yu (游智皓) is an Associate Engineer in the Environment Sensing and Control Technology center of Network and Mutilmedia Institute at Institute for Information Industry, Taiwan. He received the M.S. degrees while in the Department of Computer Science and Information Engineering at National Taiwan Normal University, in 2005.

Chun-Chin Sy (絲群欽) is an instructor in the Department of Electronic Engineering at National United University, Miaoli, Taiwan. He received the B.S. degree while in the Department of Mathematics at Chung Yuan Christian University, Chungli, Taiwan in 1975, and the M.S. degree while in the Department of Computer Science at Mankato State University, Mankato, Minnesota in 1986.

Chiu-Yang Chang (張秋陽) is currently an instructor in the Department of Electronic Engineering at National United University, Miaoli, Taiwan. He received his B.S. degree from the Department of Mathematics at Chung Yuan Christian University, Chungli, Taiwan in 1975, and his M.S. degree from the Department of Computer Science at East Texas State University, Commerce, Texas in 1986.