**Invited Paper:**

# Tiered Architecture for Remote Access to Data Sources

Karina M. Cenci Leonardo de Matteis and Jorge R. Ardenghi
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, Argentina
kmc@cs.uns.edu.ar, ldm@cs.uns.edu.ar, jra@cs.uns.edu.ar

***Abstract*—Teamwork is benefited by the use of shared data sources. Also, ever increasingly, organizational work depends on the activities of team members situated in different physical locations, including both employees who work from their homes and others who have been temporarily transferred to another place. Since, for all these reasons, accessing data remotely is a growing need, organizations implement internal systems in order to control shared data access according to user privileges. In this regard, the cost of resource transportation needed to generate communication must be considered. The main contribution of this paper is the extended reference layered architecture ICDFS-CV (Interface Control and Distributed File Systems - Communication Versioning). It allows to build a solution that, facilitates documents download and the creation and concurrent modification by multiple users through versioning control.**

**Keywords**: Distributed File System - Remote Access - Distributed Systems

## I. Introduction

Resource utilization from different locations is increasing due to the growth of mobile devices, broadband networks and wireless connections. New technologies influence organizations to ask for new requirements to improve quality and efficiency of staff work.

The access to remote resources, especially to data sources, has led to a variety of studies [9], [5], [7], [8]. Weissman et al [9] propose a paradigm called *Smart File Object* (SFO) to achieve an optimal performance in the access to remote files. This proposal uses the concept of file as an object type to provide high level interface and make use of object concepts to invoke operations and file properties. Another option, the file system *Trellis* [8], provides an abstraction to access data files using names based on *URL* and *SCL*, and its basic functionalities are implemented in user space. One of the features that it includes is the transparent access to any remote data, an important ability to metacomputation and *grid computing* areas.

On the other hand, the administration of shared data inside an enterprise or organization is commonly performed through a distributed or net file system, with capabilities to manage users in a local network or its own cloud. In some cases, these systems do not offer security, scalability and effectiveness when run through organizational boundaries.

The utilization of a virtual private network (*VPN*) [4] is an alternative, but in some cases, it is not an acceptable solution, since it does not respect internal information security policies. It is important to point out that it is not recommended to give all users access to the organizational network through *VPN* due to several security reasons. Besides, the implementation of this type of access requires the consideration of performance factors such as the speed of the available connection and the CPU usage (both the client and the *VPN* server) according to the encoded scheme of the adopted protocol (PPTP, L2TP/IPSec, OpenVPN, etc.).

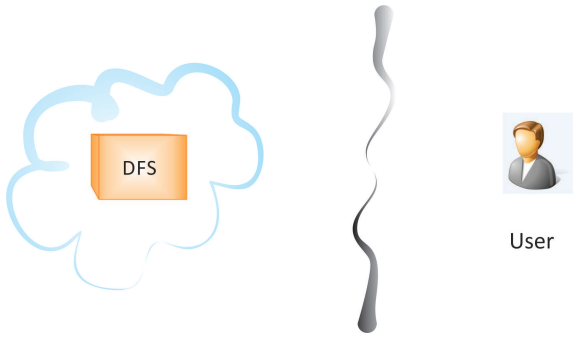


Fig. 1. Problem

Miltchev et al [7] establish a framework to compare different distributed file systems. There, they present the necessary characteristics for the comparison: authentication, authorization, granularity, autonomous delegation and revocation. These benchmarks allow an understanding of the intermediate solutions to access shared data. The relevance of the selected topics for the comparison is that remote access requires credentials management, authorization and permissions to allow the access, in this case, file access.

Section II describes characteristics of the problem to solve. Section III presents the ICDFS reference architecture for the design of a solution to this situation. In section IV an extended version of ICDFS is introduced, with the principal inclusion of a new component: the Communication Versioning. Section V presents an application example using the extended architecture to solve the problem. Section VI describes existing proposals to access shared data files of common use in some organizations. Section VII highlights advantages and drawbacks to this proposal and related works. Finally, the conclusions and future works are presented in section VIII.

## II. Problem / General Considerations

The remote access tendency is growing and the need to have high quality access means through different technologies is essential. This situation is shown in figure 1. The Distributed File System (DFS) is placed on the left side, inside the organization; while the user is seen on the other side, out of the boundaries. In this case, any employee with permission to work outside the organization and with the necessity to access the internal files (documents) is considered a user.

Organizations design and implement their own infrastructure to connect the different machines and devices.


Fig. 2. Conceptual Model

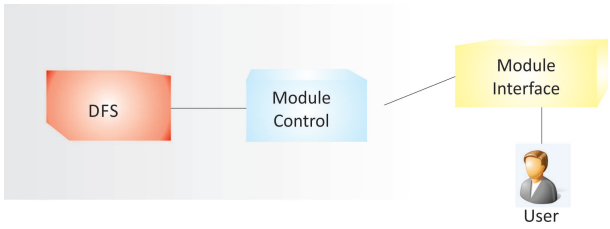


Fig. 3. Model ICDFS - Reference Architecture

Each employee has rights and restrictions to use the resources. For this particular problem, a file is considered a resource. Sometimes employees, whose tasks may require utilization of information (documents) saved in the internal network, may need to work outside the organizations or enterprises. Figure 2 shows a model in which the enterprises (organizations) and users (employees) are connected through the bridge. The inclusion of this component allows users to access the internal resources from remote locations.

The conceptual model introduces the idea of using a bridge such as a gateway. This gateway is an open point in the border of the enterprise. The challenges are to maintain all the internal information security policies and to provide efficient access from different environments.

## III. ICDFS Reference Architecture

A reference architecture to this problem was introduced in [1]. This model called Interface, Control and Distributed File System (ICDFS) is shown in figure 3.

The ICDFS is organized by tiers. Each tier is independent and they communicate among themselves through well-defined interfaces, in such way that if the behavior of the functions is changed, it is not necessary to modify the rest of the components.

The main components are: distributed file system (DFS), control module and interface module.

- Distributed file system (DFS): this component is inside the limits of the organization. It includes all the operations for managing files, accessing capabilities, sharing directories and files, managing users and providing permissions.
- Control Module: this component connects the interface module with the distributed file system, and it is the entrance door to the organization from the outside. It includes functions to guarantee access security, allowing users to gain admission to the same information that the DFS component grants. In addition, the component contains functions to read, copy, modify and add documents to the DFS.
- Interface Module: this component runs in each of the remote access points, such as cellular phone, tablet, notebook. All the operations on the DFS are done through the control module.

## IV. ICDFS-CV Extended Architecture

In this work, an extended reference architecture based on ICDFS is introduced. The relevance of this extension is the inclusion of a new compoment named Communication Versioning. Figure 4 shows the extended version ICDFS-CV.

The control module is the main component of this architecture. It is formed by the following internal ones:

- Authentication: This component controls the access to the system. It checks the user and password through the DFS module.
- Communication: This component connects with the same element of the interface module. It receives the messages with the requirements and sends the answer.
- Operation: This component contains the read, write, and modify operations. It receives the request from the communication component, and connects with the communication versioning module and DFS module.
  - Read: On-line policy is an option to implement this operation. The user needs to be connected with the remote modules during all the reading time. Another option is Downloading for the implementation of this operation. This last policy allows several users to share the file without special controls.
  - Write: this operation is equivalent to the creation of a new document in the file system. The name of the file is checked since it could not have the same name as an existing file.
  - Modify: this operation is used to change an existing file. For the implementation of this function, one of the following policies may be selected.
    - Session semantics: when a file is modified by several users at the same time, the last copy to be uploaded is the one that is saved in the system.
    - Versioning semantics: in this case, all the versions of a file are saved with the same name but with some distinctive attribute, such as the user, date and time of the uploaded file or an internal identifier.

    In this proposal, the selected policy is versioning semantics. For this reason, a new component, Communication Versioning appears in the extended architecture.

The communication versioning manages the file modifications. It connects with the DFS and the operation component.

This model shows that the control module communicates via the operation submodule with the versioning module that works as the interface with the distributed file system. In the future, interfaces will be implemented within the versioning module to establish communications with various existing protocols: Samba, SFTP or FTP.

All these protocols have operations on files and directories:

- change directory
- list a directory
- read a file
- upload a file.

The versioning module will be responsible for the resolution of conflicts according to the versioning policy. Thus, if naming conflicts arise despite the control module's

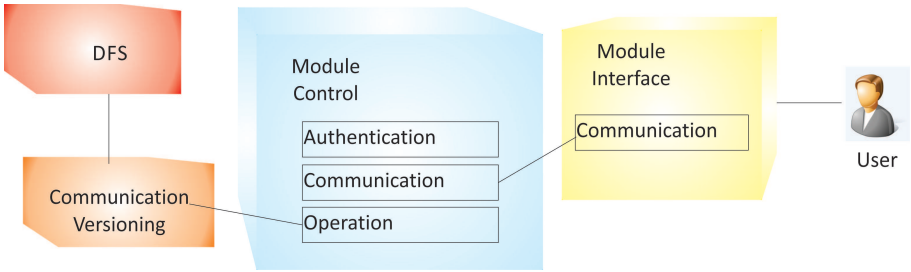


Fig. 4. Model Extended ICDFS-CV

checks (filter/safewards) for the write operation of existing files, they can be considered and solved.

In this sense, the following conflict scenarios associated with the write operation include:

- A file already exists with the same name prior to the operation, despite previous checkup at the control module.
- During a file upload process, another user creates a file with the same name and writes before completing the reception.
- During the file upload process another process deletes the immediate previous version of the file. In this case, two different actions can be taken either to follow the policy of incremental versioning in the order prior to deletion, or delete the file.

To solve the problems posed by the stated situations, a new layer, related to the control module, will be defined. Completely independent, it is implemented as a small process running on the file server itself. A new functionality provided by this process consists of the reception of file manipulation from different applications.

The defined operations of the new layer are shown in table I.

Let us now consider the operation to save a new file in the distributed file system:

- The control module defines a unique identifier for the new file to save, under the name file_id. Then, the module proceeds to contact the versioning layer through the operation file_push_begin(file_id).
- When the versioning module receives the command file_push_begin(), it proceeds to queue the filename it received for a finite period of time, and then it begins receiving the file. After the transfer upon the filesystem (with the corresponding file_id), the versioning module starts waiting to receive the command: file_push_end(file_id, filename).
- Upon receiving the file_push_end() command, the versioning module renames the received file according to the defined versioning policy. To complete this action the following steps, defined in the versioning algorithm to implement, are comp leted:
  1) Take received temporary file name, in this case: file_id.
  2) Take <filename> as prefix for the destination file.
  3) Define the new file name suffix based on the latest version of the file on the server. For example, if the file exists: filename [(version_number)] = version_number suffix (where version_number belongs to the set of positive integers).
  4) Rename the file as: filename(suffix+1).
- If communication between the control module and versioning layer is broken, the last component will delete those temporary files created after a fixed time.
- If the versioning layer could not receive the file or could not complete the renaming operation according to the versioning policy, it will notify the control module with the corresponding error message. Also, the temporary file generated during the transfer process (reception via file_push_begin()) will be deleted.

Figure 5 shows the steps required for a successful write operation and the messages exchange among the modules.

In the communication between defined modules schema, any error is only reported and the necessary measures are taken to remove temporary data, but fault tolerance does not imply automatic retry attempts. The user will be informed of the errors produced, and he shall take the corresponding/necessary actions to retry writing, reading or achieving the desired modifications.

## V. Application Example

The proposed reference architecture is used to design an architecture for the implementation of a remote access system in a distributed repository located in the internal net of an organization.

The following requirements for the implementation have been considered:

- Remote access to the common files of the organization.
- Respect for the same access permissions in folders and files provided by the DFS.
- Access in read mode to the files as the principal functionality.
- Files and folders not to be deleted.
- Write access as a secondary functionality.

The environment in which the example is applied has some characteristics. First, the shared files are placed in the repositories, which are accessible through SMB (*Server Message Block*) protocol over Microsoft AD (Active Directory) services of Microsoft Windows 2003R2.

Second, different area users may read those folders and files they have access permissions through the internal policy granted by the organization. In addition, a remote access service to the internal documents is an important and essential functionality to the users. This access may guarantee the same policies and security measures that apply in the internal net.

The chosen policies of the operation component (Control Module) are the following:

1) read operation: it allows the downloading of the file to the device;
2) write operation: it does not allow the creation of a new document with the same name of another document in the corresponding folder;
3) modify operation: the versioning semantics is chosen. This decision is based on the idea of providing

| Operation | Description |
|---|---|
| file_push_begin(<file_id>) | used to begin the operation of creation of a file (new one or new version). |
| file_push_end(<file_id>, <filename>) | used to saved the file in the DFS. This function checks the versioning. |
| file_get(<filename>) | used to obtain the file. |

TABLE I
COMMUNICATION VERSIONING: OPERATIONS

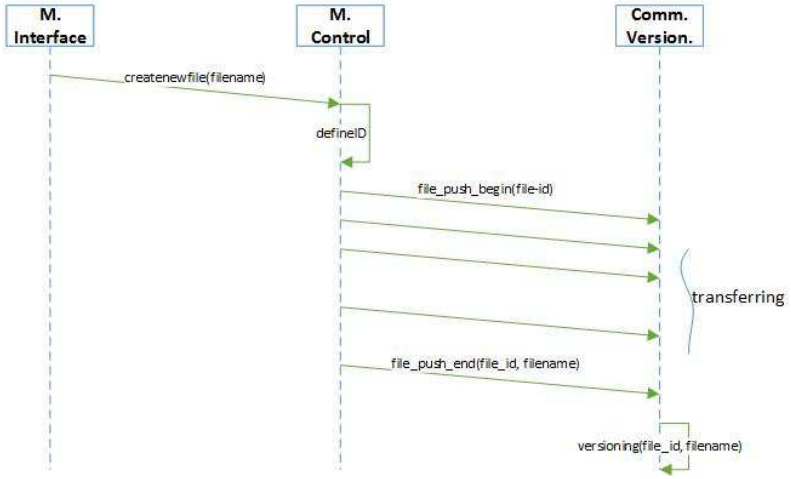


Fig. 5. Successful write operation

users with all the modifications made concurrently in a period of time.

### A. Components

For the implementation of a service environment that meets the functional requirements specified in the previous section the following components were used:

- Virtual machine with GNU / Linux CentOS 6 operating system.
- Samba's smbclient component.
- A web server with Nginx.
- Use of PHP-FPM to interact with the web server.
- Scripting service through PHP-CLI module with the core of PHP in version 5.4
- Module ngx_http_ssl_module to provide HTTPS support.
- OpenSSL library.
- Control module implemented in PHP language.
- Web interface implemented in PHP language.
- Mobile devices with embedded web browsers.

The component selection was guided by their advantages for the purpose of the formulated setting. Thus, PHP is a scripting language, currently listed as a general purpose language for running in web servers.

In the implementation, the modules developed in PHP run through an FPM interface that allows higher speeds and better performance compared to those that could be achieved with a deployment using Apache web server and PHP as one of its modules.

Meanwhile, PHP-FPM (FastCGI Process Manager) is a FastCGI interface and it is also an alternative implementation with additional features that make it suitable for use in web applications of any size but with high amount of requests per time unit.

Furthermore, FastCGI is a protocol that works as an interface with other programs that communicate with a web server. The main objective of this protocol is to reduce the additional time used in communication with the web server, allowing it to service a greater amount of requests simultaneously.

To provide secure communications between the interface module and the control module, a communication scheme based on the HTTPS protocol was chosen, so the objectives of data confidentiality policies of the organization are achieved. Besides, directives HSTS (HTTP Strict Transport Security) are used to achieve secure access to the interface module.

## VI. RELATED WORK

Among the alternatives to manage access to shared data the use *HFS*, mod_dir of Apache and WebDAV may be considered.

In the first place, *HFS* (HTTP File Server) [6] allows an easy file sharing among a workgroup through *HTTP* protocol. File sharing may be limited to a group of users or allow everyone to access them. The difference between this and other file systems is that it does not require the net. *HFS* is a web server, characteristic that allows files publication through a website presentation. *HTTP* protocol has weaknesses in security aspects, since the traffic is sent in plain text and every sent data bit between the web server and the client can be intercepted and read by all the machines that are in the way to the final site. This tool works over Microsoft Windows operating systems. One disadvantage of this is that it does not support an authentication scheme through *ADSI* (Active Directory Services Interfaces) interface.

A second option, *Apache Module* mod_dir [2] is used to redirect the trailing slash and it is used as the file directory index. It is a simple way to share files, especially those used to access free distribution data sources. One advantage of this tool is the availability to install it over operating systems that can execute Apache HTTP Server such as: Microsoft Windows, GNU/Linux, Unix, OS X, etc.

Another possibility is *WebDAV* (Web-based Distributed Authoring and Versioning). It is a set of HTTP extensions,

| Criterion | ownCloud | I-ICDFS-CV |
|---|---|---|
| Authentication | $\surd^{1}$ | $\surd^{2}$ |
| Privacy | $\surd$ | $\surd$ |
| Auditing | x | $\surd$ |
| Reading | $\surd^{3}$ | x |
| Writing | $\surd^{4}$ | $\surd^{5}$ |
| Modification | $\surd$ | x |
| Simultaneous Modificacion | x | $\surd$ |
| Downloading | $\surd$ | $\surd$ |
| Verioning | $\surd^{6}$ | $\surd$ |

1: LDAP - 2: LDAP / API ASDI
3: Only edited files in collaborative mode - 4: Preliminary Check
5: Control Name - 6: Manual
TABLE II
COMPARISONS

that allows users to cooperate among them to edit and manage files in web servers through the net. *WebDAV* is documented in RFC 2518 and extended in RFC 3253. RFC 2518 specifies the set of methods, headers and secondary content types to HTTP/1.1 to manage properties, create and administer collections of resources. For common users, *WebDAV* allows web team developers and other workgroups to use a remote web server as easily as a local file server.

Hernández and Pegah [3] propose a solution to shared files using *WebDAV*. *LDAP* (Lightweight Directory Access Protocol) is included in the system to maintain a unique registration. This extension allows a shared access service without interruption. All these components are integrated through the Apache web server that allows the usage of *WebDAV* extensions and the meta-directory model LDAP for users authentication. The advantage of this implementation is that it gives a solution compatible with *NFS* y OS X.

## VII. COMPARATIVE ADVANTAGES

In the proposed architecture, the authentication can be defined on different implementations of distributed files that use *LDAP* or some other type of system to catalog users and objects with different permissions, since the control module is responsible for the remote users authentication. The authentication depends on the underlying system that provides that service. So whether a validation on each requirement is needed to make —as in the presented implementation example— or if an enduring validation is obtained for a certain time —in the case of using the API ASDI, for example—, the control module can be adapted to both schemes. Another advantage of this permission validation scheme is that, if the administrator modifies permissions to the objects in the distributed file system, they immediately produce an effect on the files that the user may or may not access.

This proposal is compared with a recognized and widely used application, such as the ownCloud system. This system, also implemented in PHP with interfaces to access different file systems, does not have a synchronization layer for the read/write accesses. In ownCloud, simultaneous access to rewrite the same file by two different users simultaneously causes only the most recent changes to remain available making it to fail. These problems occur because smbclient is used as wrapper and access requests are not previously enqueued as in our proposal. The system ownCloud implements the report errors policy if they occur as a result of executing the call to smbclient.

No locking mechanism on the access for writing a file is observed at the ownCloud source code, it only performs a preliminary check for its existence and asks the user for the policy that should be adopted with respect to the overwriting or writing of a new file with a number as suffix to indicate the file version.

Moreover, the architecture provides access through the graphical interface that is presented to the user on any mobile device with a browser with support for *HTTPS* protocol and *HTML* language. The goal is to provide universal access to all client computers without installing a specifically compiled application for each architecture and operating system available today in widespread and massively used devices. Users are not created on this architecture: they already exist on the file sharing scheme the organization has.

Besides, this way, basic safety requirements are ensured: authentication, privacy and auditing. With respect to the first two, they are inferred from the previous description, but the third requirement, the audit, is an advantage of the architecture, as administrators of the organization may have a record of all events on access to internal company files from the outside and may account for authentication, downloads and modifications made by each user. Note that without a control module that provides these services, related system proposals presented above do note make available all the information necessary for a complete audit, unless the system administrator changes specific compartments of these systems to include this type of audit, either by modifying policies of the specific distributed system (eg, in the case of implementations on Microsoft AD) or by mod_dir logs in Apache, to name some particular cases.

Table II shows a summary of criterions comparison of this proposal and ownCloud.

Another advantageous feature of is the fact that the internal network is not exposed completely to the remote computers users employ to access, as in the case of the use of virtual private networks (VPN). This proposal allows organizations that do not implement VPNs as a service for all users, because of internal policies, to provide this service more generally without the VPNs' security inconveniences. In these organizations, only those users with exceptions and special privileges may use the VPN service.

## VIII. CONCLUSIONS

The media insertion in society changes the way people perform daily activities, for example, using the Internet to make payments, inquiries and reservations, or accessing government services, etc. In enterprises, network connectivity allows employees to carry out their work activities from different locations. In order to do so, an option

is the use of *VPNs*, which have both advantages and disadvantages.

As an alternative, this work proposes ICDFS-CV, an extended reference architecture to model the access to data sources located within the organization's boundaries. This alternative ensures that each outside user gets the same capabilities and permissions as if he were working within the physical boundaries of the organization. Moreover, it includes a special component to manage documents versioning. Benefits are associated with the auditing of operations executed from the organization outside.

As future projections, this work proposes the incorporation of submodules into the control module to access different distributed file systems, along with the authentication feature on various systems with LDAP as a related service, or any other possible validation schemes. Therefore, it aims at specifying a general framework specification for the control module as a relevant goal, so that it, in turn, can incorporate other sub-modules to the necessary extent, in order to broaden the proposed reference architecture's spectrum of use.

## REFERENCES

[1] K. Cenci, L. de-Matteis, and J. Ardenghi. Arquitectura en capas para acceso remoto. In *CACIC 2013*, 2013.

[2] The Apache Software Foundation. *Apache Module mod_dir*, 2013. httpd.apache.org/docs/current/mod/mod_dir.html.

[3] L. Hernández and M. Pegah. Webdav: What it is, what it does, why you need it. In *SIGUCCS '03, ACM*, pages 249–254, 2003.

[4] R. Hills. *Common VPN Security Flaws*, 2005. http://www.nta-monitor.com/.

[5] T-J Liu, C-Y Chung, and C-L Lee. A high performance and low cost distributed file system. In *Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on*, pages 47–50, 2011.

[6] M. Melina. *HFS Http File Server*, 2002. http://www.rejetto.com/hfs/.

[7] S. Miltchev, J. Smith, V. Prevelakis, A. Keromytis, and S. Ioannidis. Decentralized access control in distributed file systems. *ACM Comput. Surv.*, 40(3):10:1–10:30, August 2008.

[8] J. Siegel and P. Lu. User-level remote data access in overlay metacomputers. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'02)*, pages 1–4, 2002.

[9] J. B. Weissman, M. Marina, and M. Gingras. Optimizing remote file access for parallel and distributed network applications.