Distributed Virtual Cloud File System

David (Wei) Jia

6.UAP Report, 2011

Professor Nickolai Zeldovich

PDOS Group, CSAIL

Index

Introduction

Problem

Web users increasingly store and juggle their files across multiple locations sharing between web services like Facebook, Google Doc, company file shares and their personal laptop. There is no easy way to get data out of the cloud and share it across these services. However, most cloud providers expose APIs, which are typically well documented but also unique to their implementation and thus not interoperable. Furthermore, while applications that allow users to authenticate with different web services and access multiple web APIs and platforms are constantly being developed, there is no centralized way to access these data. Each individual application developed requires the developer to access each of the data points (such as the web applications listed above) separately. If there are M applications being developed and N data sources to integrate, this becomes an O(N * M) problem, similar to the initial phase of the World Wide Web. The Web has been innovated and has eventually become a client-server architecture model. However, this has not yet been done for data. With the emergence of linked-data, web services, in theory, are able to publish their data publicly through special relational databases that are linked with one another. However, as with most web standards, adoption rates among services are slow. Even though most web services utilize a standard API protocol method such as REST, most services' data do not link with one another. We propose a solution to these problems, which will be described in the current paper.

Motivation and Goals

We propose a Distributed Virtual Cloud File System (DVCFS), as seen in Fig 1, that integrates these open APIs and centralizes data into a single virtualized file system and allows user to interact with multiple cloud services and web apps as well as their personal computers. DVCFS uses a virtual file system architecture that allows users to easily connect to any external file system to access and share their files across multiple cloud services. DVCFS will create a centralized location for data access that connects different data points based on user information. The DVCFS will centralize all data points for a particular user. This will not only allow individual users to have control of their data in a hierarchical sense, but will also allow third party application developers to organically and robustly develop applications that span data access across multiple sources.

Design

Architecture

Fig. 1 shows a high level architectural design of the DVCFS. User applications on the client side communicate and transmit information and data to the DVCFS through a standardized protocol that the DVCFS enforces. The DVCFS handles client requests to read or perform operations on a particular file system by using the file system handler corresponding to the given file system that the client requests to access. The DVCFS uses the file system handler to communicate with the remote file server through the specific protocol of the remote file server. The file system handler translates the standardized protocol enforced by the DVCFS into the specific protocol of the target remote file server. The remote file server performs the given action and returns the response to the file system handler of the DVCFS. The file system handler of the DVCFS then translates the specific protocol of the target remote file server into the standardized protocol enforced by the DVCFS.

Robustness and Generalizability

Fig. 2 illustrates an exemplary process for adding novel file systems to the DVCFS. In Fig. 2, a user first triggers an event to add a particular file system given a set of credentials the user inputs in the client user interface. The given file system is converted to a file system ID and passed to the DVCFS along with the user's credentials. The DVCFS resolves the file system ID into a file system handler to perform the add file system operation. The DVCFS then loads the file system handler libraries. The file

system handler next checks the credentials against the file system to be added. The file system handler does this by sending a message to the remote file server with the user provided credentials. The remote file server returns to the file system handler whether or not it was able to authenticate with the given credentials. If the remote file system was not able to authenticate, the user is notified in the client user interface and may be asked to try again. If the remote file system was able to authenticate, the file system handler checks the database for whether the given file system and credentials already exist for the user. Whether the file system credentials exist in the database for the user is returned to the file system handler. If the file system credentials already exist for the user, the database is updated with the new credentials for the given file system and the user is notified. If the file system credentials for the user do not yet exist in the database, the credentials for the given file system are saved for the user and the user is notified.

User Action Flow

Fig. 3 demonstrates an end to end process flow of performing operations on the DVCFS. In Fig. 3, the user, the interaction of the user with the user interface, the client, the user interface on the client side, the action of sending requests from the client to the DVCFS, the main request receiver, the file system handler, the action of getting saved credentials, the database and cache, the action of retrieving saved credentials, the action of connecting to the target remote file server, the remote independent file system, the action of performing the specified request and returning the result to the DVCFS, the response parse engine, and the action of standardizing the response and returning it to the client.

The user interacts with the user interface on the client to perform an action on a certain file system. The request is sent by the client to the DVCFS through a standardized protocol enforced by DVCFS. The main request receiver in the DVCFS receives the request sent by the client. The main request receiver passes the request to the file system handler for the given file system that the user wants to perform operations on. The file system handler checks in the database for the user's saved credentials for the given file system. If exists, the database returns the saved credentials to the file system handler. The file system handler connects to the target remote file system and sends the request in the protocol specific to the remote file system. The remote independent file system in the remote file system processes the request and sends the response back to the file system handler in the DVCFS. The response parse engine in the DVCFS standardizes the response and sends the response back to the client. The user interface in the client displays the results to the user. In some cases, the file system handler may search a cache in the DVCFS for preloaded results to seek operations, although this is not a necessary component or behavior for DVCFS.

Implementation

Backend

The backend of the current implementation utilizes PHP as the main framework for architecting the DVCFS. The object-oriented setup of the PHP backend follows the high level architecture fairly closely. There is a FileSystemManager, which manages the file systems by receiving all action messages related to any file system. Along with the file system action, each action message contains the file system ID, which is unique to each data point. Each file system has a FileSystem object, which understands how to resolve standardized DVCFS file system operations to file system specific operations. FileSystemManager is in charge of taking any incoming file system operation, regardless of the specific file system. It then looks up and resolves each file system ID to a FileSystem object, which is then passed the specific file system operation and understands how to handle it specific to the file system in question.

The standard file system operations supported by DVCFS are: new directory, delete file or directory, move (including rename), and copy. These standardized DVCFS are chosen to match standard file system operations. This will also allow for easy integration of new file systems, which will be discussed below.

User systems are fairly standard for web applications. A MySQL database is used to store user information, including authentication information for each file system.

Frontend

Although for developers the user interface of the system is not crucial, how the information and data is displayed visually is very important for regular users. The current implementation of the system, as seen in Fig. 4, creates a desktop-like interface where each separate data point is mapped as a file system on the virtualized desktop. This style is chosen for a multitude of reasons. First of all, users are familiar with desktop-like interfaces. It is the GUI that is used for most major operating systems and users have an intuitive feel for it. Second, a desktop-like user interface allows for a file system hierarchy. This file system hierarchy allows users to easily manage their data through a drag and drop interface, which minimizes the number of clicks a user must go through in order to perform an action.

The frontend is built completely of JavaScript and standard HTML / CSS elements. This is to aid distribution. With a system completely on the web and accessible through any browser, the barrier to adoption is greatly dropped. There is no download or third party plugs (like Flash or Java) required. Any user with a modern browser can access the application.

Finally, a unified user interface removes the necessity for users to learn the different interfaces of separate applications. With the many applications that store data, users must learn a new user interface for each new application he or she decides to use. This can be tedious and difficult to remember. Having a unified front end interface lowers the barrier of adoption by removing the need to learn new interfaces.

New File System Integration

A new file system can be easily integrated into DVCFS. The key for integration is the FileSystem class, which is specific to each specific file system. The FileSystem class for a specific file system is the class that understands how to resolve standard file system operations and into API calls to the specific file system. This is the key interface that needs to be completed when adding a new file system to the supported file systems of DVCFS. For example, when integrating Facebook photos, the Facebook photos FileSystem class should resolve a new directory command as a command to create a new photo album.

Developers

Developers can easily build applications on top of DVCFS. The interface that application developers works with is a centralized file system. To the application developer, all data, regardless of the source, is from a single file system. The only difference is that each file has a file system ID, which identifies which real file system it's in. Thus, any external API that DVCFS offers developers would include standardized object structures for all virtualized file systems. This gives developers an abstraction layer and allows for robust and agile application development.

Use Cases and Evaluation

Possible Applications

Any ordinary application that utilizes a file system can be built on top of DVCFS where DVCFS acts as an abstraction layer for the application and the application can access files from any data location, but to the application, it seems as if all the data is from a single virtualized file system. In this section, we will briefly discuss several different possibilities for applications that can be built on top of DVCFS.

Personalized universal search: since users have all their data sources added to one single virtualized file system, the DVCFS file system, with some indexing techniques, one could develop an application that searches through all of an user's personalized data with one search. This is very convenient for the user as he or she only has to type in one search bar.

Backup: backups of file systems have been proven to be stable in various scenarios. Since DVCFS creates a virtualized file system containing virtualized data from multiple sources, it is more or less trivial to implement a backup system on top of a DVCFS file system.

Music, photo, or video management software that spans data from multiple sources: digital media in today's world is sparsely distributed across the Internet, so traditional digital media management tools like iTunes, iPhoto, and iMovie, which scan files on a single machine, makes little sense. To truly manage digital media files, software needs to be able to access data from a multitude of sources, accessible through separate APIs. DVCFS allows for such digital media management tools to be easily built

with its virtualized file system. Converting a traditional digital media management tool to one that manages files from various distributed sources is more or less trivial with DVCFS, since it virtualizes different file systems to a single file system.

Technical Evaluation

We look at two different perspectives in this section: efficiency for users, and efficiency for developers.

Users: in a recent survey given to college students around the Boston area, it was found that the average number of different locations users have data is seven. This is a large number of data sources to juggle between. Users have a hard time managing and controlling their data. Even without a full-fledged universal search function, it is found that users are able to find data across different data locations 100% faster than normal—having to dig through multiple data sources.

Developers: the creation of DVCFS allows developers to easily integrate user data from multiple data points. Without DVCFS, developers must deal with a multitude of issues. First, different data APIs have completely different implementations, often times with flawed documenting, each one resulting in extra time spent in understanding the API. Second, different data sources have different data model structures. Integrating them requires extra engineering effort to resolve their differences and similarities. Third, users must re-authenticate with each service the application developer wishes to integrate, which raises barrier to adoption and decreases the adoption rates of the finished application. DVCFS fixes all of these problems. DVCFS also solves the O(M * N)

problem mentioned earlier in this paper whereby each of M application developers must learn how to integrate each of N different data storing services.

Reception

The current DVCFS implementation has received very positive public reception. It is being used by more than 300 members of the MIT Media Laboratory. The technology was the winner of the MIT $100K Entrepreneurship Competition Web/IT Track and the MIT Linked Data prize.

Conclusion

A Distributed Virtual Cloud File System (DVCFS) allows for access to data from multiple file systems through a single protocol. The user is able to add file systems by providing credentials to be saved. Through the client, the user is able to perform an action on a certain remote file system the user has added. To do this, the user performs the desired action through the standardized protocol enforced DVCFS. The DVCFS then finds a file handler whose purpose is to interact with the remote file system. The file handler then transfers the standardized protocol into a protocol specific to the remote file system. The file handler then asks the remote file system to perform the desired action. The result from the remote file system is then re-standardized by the file handler to be compatible with the DVCFS, and the result is then returned to the user on the client side. The DVCFS allows for robust third party applications to perform file and data operations on data from distributed sources. The extensibility of DVCFS allows the file systems that the DVCFS supports to be extended to support more file systems.

Figures

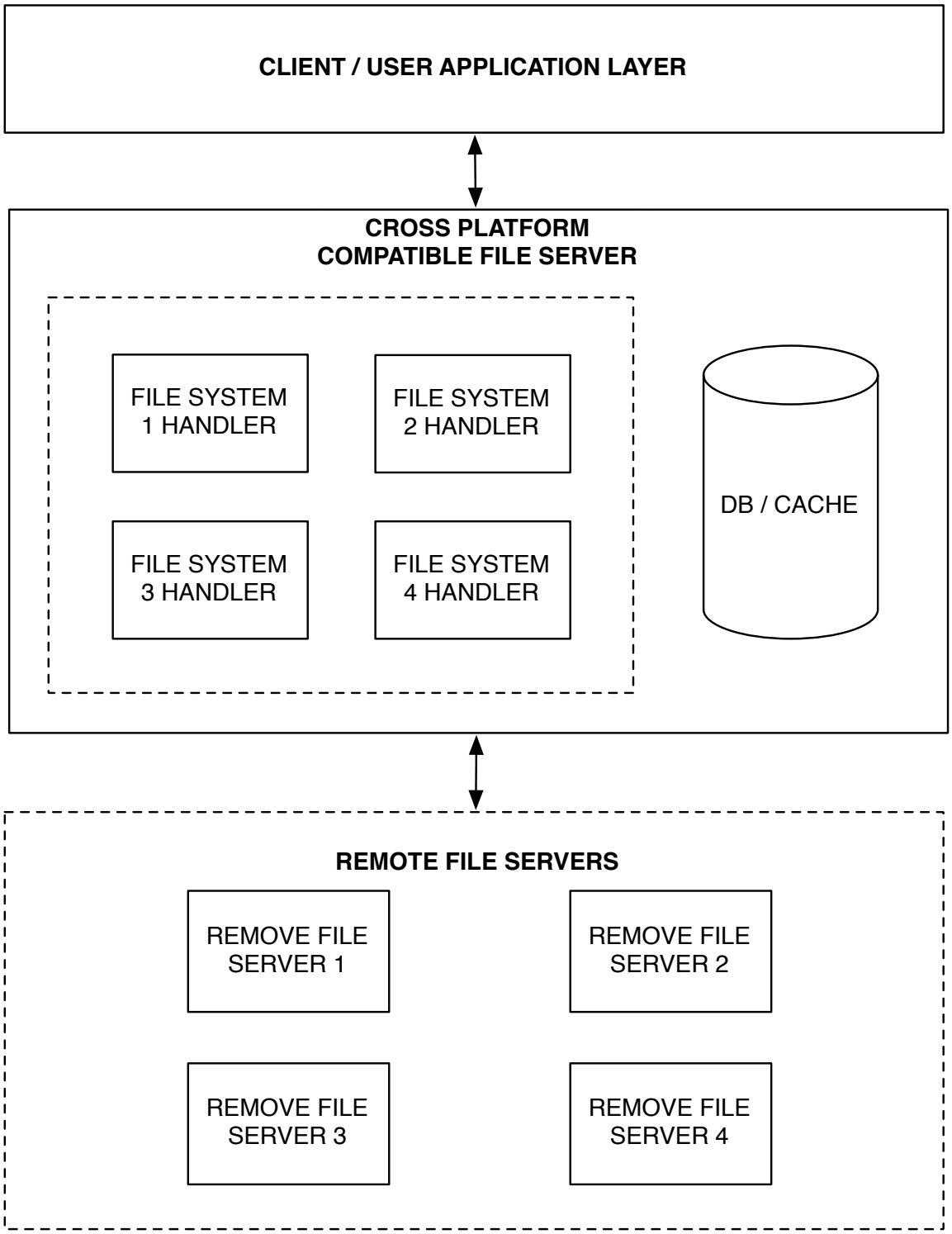

FIG 1.

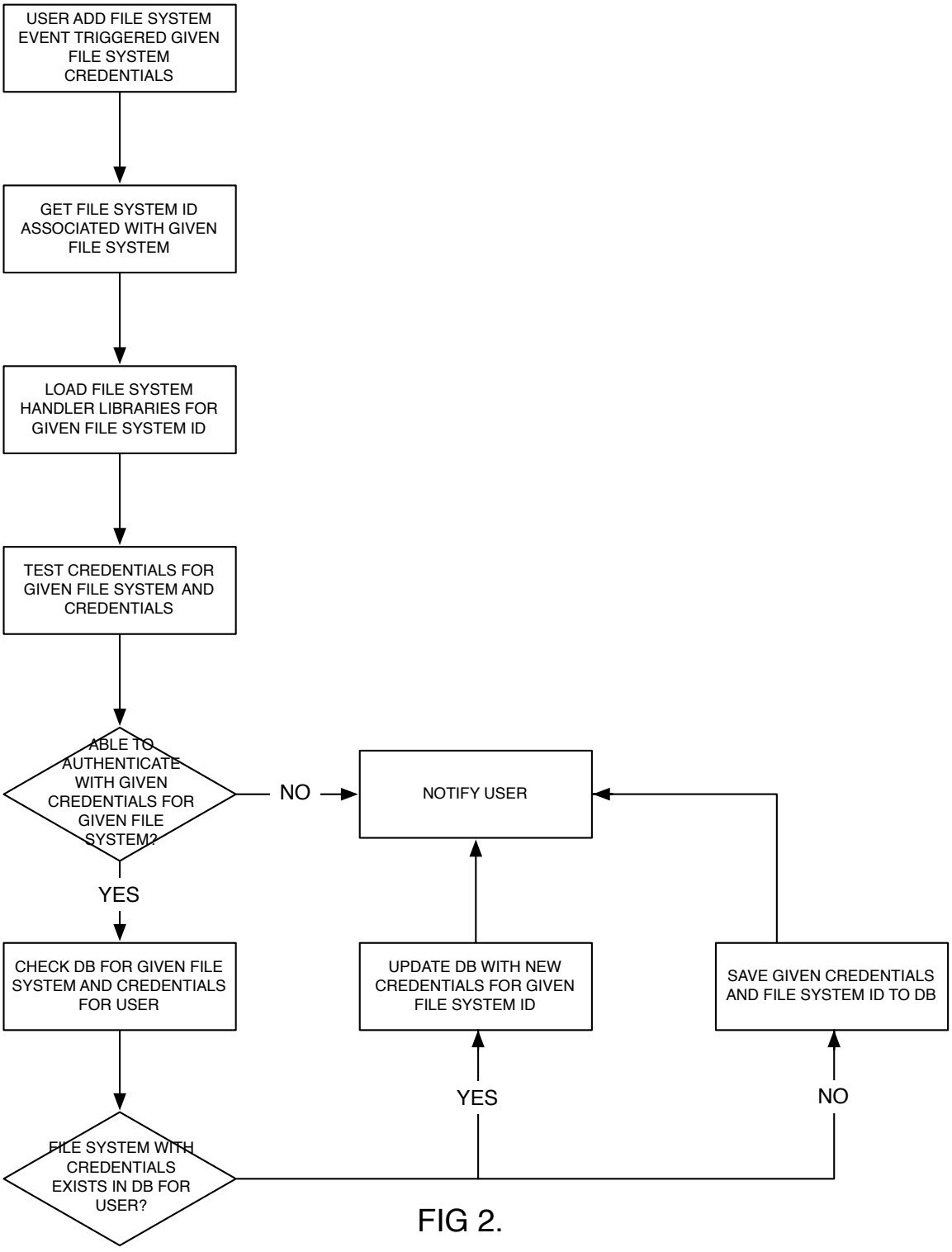
USER ADD FILE SYSTEM EVENT TRIGGERED GIVEN FILE SYSTEM CREDENTIALS
GET FILE SYSTEM ID ASSOCIATED WITH GIVEN FILE SYSTEM
LOAD FILE SYSTEM HANDLER LIBRARIES FOR GIVEN FILE SYSTEM ID
TEST CREDENTIALS FOR GIVEN FILE SYSTEM AND CREDENTIALS
ABLE TO AUTHENTICATE WITH GIVEN CREDENTIALS FOR GIVEN FILE SYSTEM?
NO
NOTIFY USER
YES
CHECK DB FOR GIVEN FILE SYSTEM AND CREDENTIALS FOR USER
UPDATE DB WITH NEW CREDENTIALS FOR GIVEN FILE SYSTEM ID
SAVE GIVEN CREDENTIALS AND FILE SYSTEM ID TO DB
YES
NO
FILE SYSTEM WITH CREDENTIALS EXISTS IN DB FOR USER?

FIG 2.

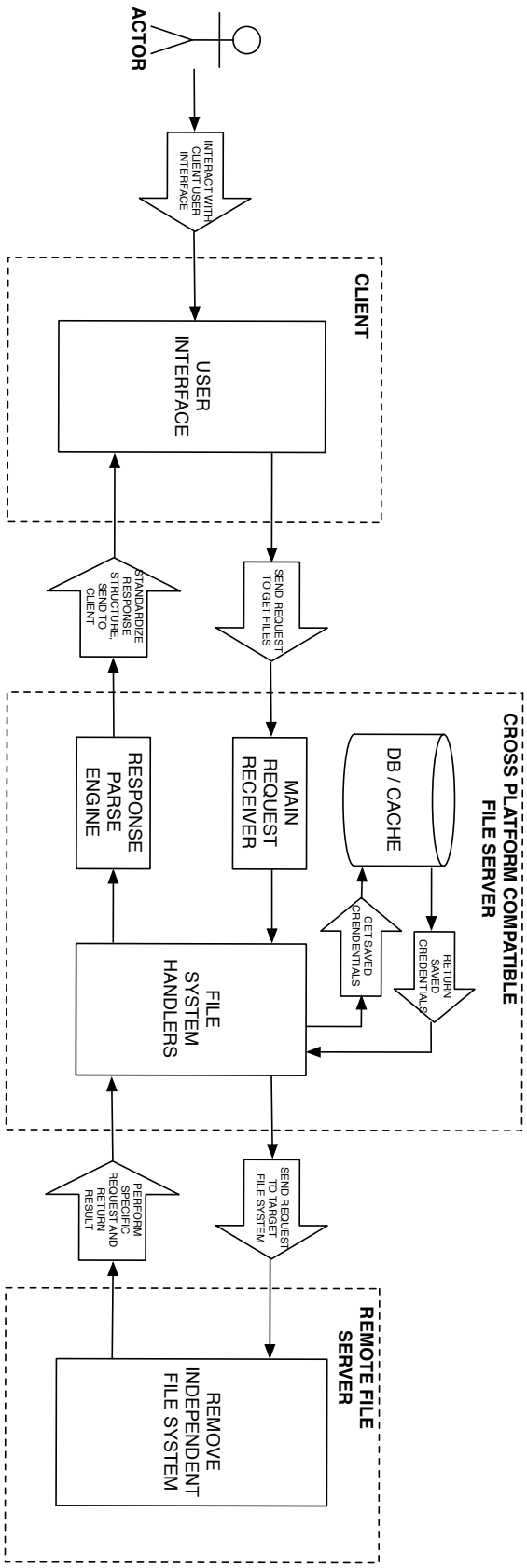
ACTOR
INTERACT WITH A CLIENT USER INTERFACE
CLIENT
USER INTERFACE
SEND REQUEST TO GET FILES
STANDARDIZE RESPONSE STRUCTURE SEND TO CLIENT
CROSS PLATFORM COMPATIBLE FILE SERVER
MAIN REQUEST RECEIVER
RESPONSE PARSE ENGINE
DB / CACHE
GET SAVED CREDENTIALS
RETURN SAVED CREDENTIALS
FILE SYSTEM HANDLERS
SEND REQUEST TO TARGET FILE SYSTEM
PERFORM SPECIFIC REQUEST AND RETURN RESULT
REMOTE FILE SERVER
REMOVE INDEPENDENT FILE SYSTEM

FIG 3.

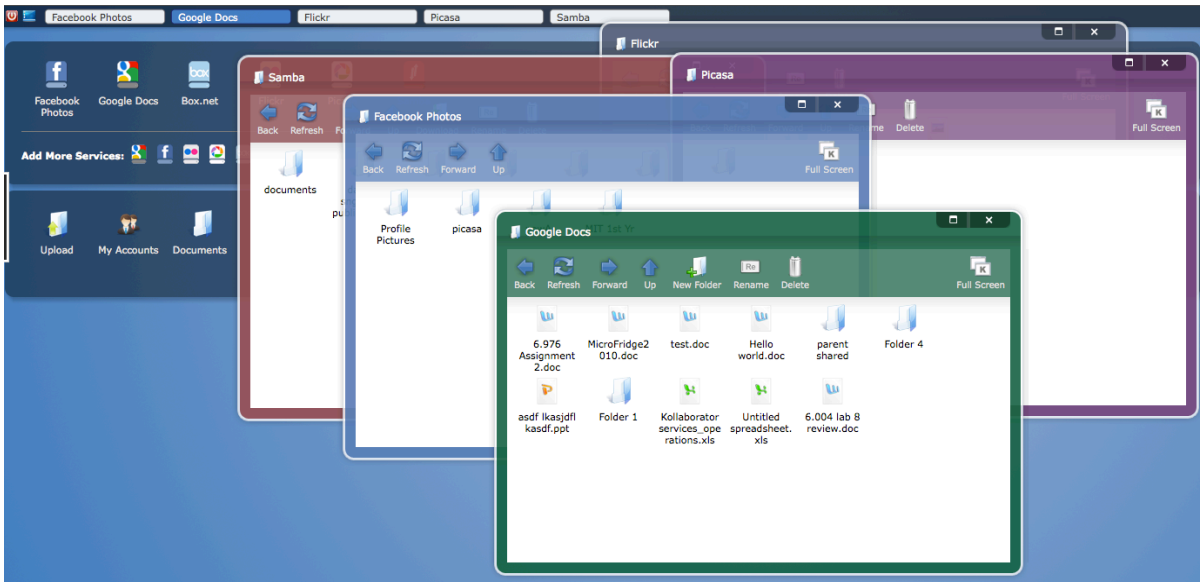
Facebook Photos
Google Docs
Flickr
Picasa
Samba
Box.net
Add More Services:
Upload
My Accounts
Documents
Back
Refresh
Forward
Up
New Folder
Rename
Delete
Full Screen
documents
Profile Pictures
picasa
6.976 Assignment 2.doc
MicroFridge2 010.doc
test.doc
Hello world.doc
parent shared
Folder 4
asdf lkasjdfl kasdf.ppt
Folder 1
Kollaborator services_ope rations.xls
Untitled spreadsheet. xls
6.004 lab 8 review.doc

FIG 4.