Efficient Synchronization of files between Digital Safes

[1]Vani H U , [2]Swathi V S,[3]Spoorthi U K, [4]Shwetha S, [5]Namitha A R

[1234]UG Students, [5]Assistant professor

Department of Computer Science and Engineering

BGS Institute of Technology , BG Nagar

Abstract-Cloud storage offer the availability to the end user. Thus, addressing the mobility needs and device's variety has emerged as a major challenge. Data should be synchronized automatically and continuously when the user switch from one device to another device. The owner should be offered the service of sharing the data with any specific user by the cloud service.

The goal of this paper is to develop a secure framework that fortify file synchronization with high quality and least possible resource consumption. As a step towards this intention, we advance the SyncDS protocol with its associated architecture. The synchro-nization protocol efficiency raises through the choice of the used networking protocol as well as the strategy of changes detection between two versions of file systems located in different devices. Our try out results show that acquiring the Hierarchical Hash Tree to notice the changes between two files and validate the WebSocket protocol for the data exchanges improve the efficiency of the synchronization protocol.

Index Terms-File synchronization, Digital Safe, HTML5 Local Storage API, WebSocket, Hierarchical Hash Tree, Web services.

I. INTRODUCTION

With the invasion of used smart objects, the cross-device data management remains the concern of multiple research and industrial works. Several Cloud storage solutions are proposed to handle this issue. However, the commercialized products are usually based on proprietary solutions. They obviously lack transparency for the users how their data are managed in the client and Cloud side. These closed solutions depend heavily on the used machine. Cloud file syncing's popularity has risen due to number of employees working remotely, telecommuting, or travelling, who need access to certain files. To meet this need, businesses turn to cloud file syncing services. Like typical cloud storage, cloud file syncing services work in either public or private environments. Hybrid environments are also an option.[1]

+Adding the probative value into the Cloud storage tends to guarantee the proof of the storage actions performed by the users. A standardized Cloud storage solution with the proba¬tive value is introduced by the Safe Box As A Service (SBaaS) [1]. It is a standardized architecture that provides a secure environment to store sensitive documents. The conception of this safe follows the AFNOR specifications [2]. It is a standard that offers the best possible security, integrity and quality to preserve user's data. It is characterized by its probative value as a proof of storage is preserved in a third trusted party.

Introducing the synchronization in a Digital Safe platform implies the definition of a Client Digital Safe. In our platform, the client safe is based on HTML5 APIs. The major interest of using HTML5 is to avoid proprietary solutions and to adopt standardized one. It offers also the transparency to the end user and guaranties the mobility and portability while preserving an efficient traffic exchange. In this paper, we highlight the major need of defining a standardized architecture for file synchronization with efficiency considerations. First, we focus on the synchronization between the Client Digital Safe and the Cloud Digital Safe. Second, we address the efficiency in this context by the choice of the used networking protocol as well as the strategy of changes detection between two versions of file systems.

The rest of the paper is structured as follows. In section 2, we give an overview on data synchronization challenges. In section 3, we present the synchronization architecture. Section 4 addresses the Digital Safe synchronization protocol (SyncDS) and the changes detection needed in the synchro¬nization. We deal with the implementation in section 5. We analyze then the performances of our proposed protocol in section 6. We end up with a conclusion and perspectives.

II. OVERVIEW ON SYNCHRONIZATION PROTOCOLS

A. Efficient Synchronization protocol requirements

In the context of file synchronization, the protocol has to provide four main key properties that should be respected in order to guarantee an efficient bandwidth, the fastest data exchange and an effective computation [3] [4] [5] [6]:

-Property 1: Low computation of the data and metadata at the client side. This property adds the scalability and enables simultaneous synchronization.

-Property 2: Efficient change detection between the file system versions of the client and the server. The goal is to

reduce the time of matching and therefore, the time of whole synchronization.

-Property 3: Reducing the number of synchronized files by finding the maximum number of matched content.

-Property 4: Reducing the messages between the client and the server.

B. Detecting changes in file systems

The different proposed change detection algorithms can be classified into three main approaches (we consider that the node A intends to send the changes performed on its file system to the node B):

- operation Approach: This approach is based on recording the different operations performed on the node A and sending them to the remote node B [5]. These operations are sent to the node B to update its replica. However, storing the operation is a consumption of the storage memory, unless a log merging is adopted such as the case in[7], which reduce the used storage.

- Changes Approach: The main idea behind this approach is to save a copy of the last synchronized file system version (i.e. just before being offline). After a series of modifications and when the user becomes online, the old copy and the new version are compared. Actions are therefore detected and sent to the remote node B [8]. This approach proved its worth. However, it needs additional storage space for archiving the file system. In addition, there is a lack of automatic detection since it must be triggered periodically. Otherwise, the change detection can use operating system modules such as Inotify on Linux or FSEvents API on Mac OS X. It is obvious that the synchronization application will depend heavily on the operating system.

- Differencing Approach: The node A sends an abstract of its files and directories to the remote node. This abstract includes some metadata of files. In the case of dropbox [9], the metadata contains the object path, object type (files or directories) and the hashs of 4Mb file's blocks. In the case of Rsync [10], it sends the hash of file blocks, and Taper [3] sends the Hierarchical Hash Tree, the hash of file's chunks and the hash of file's blocks. The node B, therefore, compares its abstract with the one received from A to detect the changes. It asks A then to send back missing blocks of files. This approach cannot guarantee an automatic synchronization and sending periodically the abstract leads to an efficient bandwidth usage. In this case, even the structure of the abstract should be well chosen to guarantee a fast matching at the target. That is why we introduce the HHT into the abstract structure. Taper defines an algorithm to detect only the changes blocks in each file of the filesystem. However, our approach and proposed algorithm detect the changes in the whole filesystem with the different operations performed by the user such as, rename, copy, move, add, modify, etc.

Adopting the differencing approach leads to focus on detect¬ing the changes between the abstract sent by node A and the abstract of the node B. In the case of data synchronization, the abstract consists of a part of file system metadata. Speaking in terms of graph theory, a file system can be modeled as a tree. Therefore, the metadata of a file system can be organized to be modeled as a tree structure. A tree is a rooted and ordered graph consisting on nodes interconnected with a parent-child relationship. In the file system context, the node that has a parent is called a root which is the main directory of our file system. However, nodes that have not child are the leafs which matches to the files. Nodes in between are called inner nodes. They represent the different directories of the file system.

Many works are focusing on general problems of detecting changes between documents, mostly flat files. For example, Unix diff is one of the most popular change detection utilities that use the Longest Common Subsequence (LCS) algorithm to compare two files. We find also the Concurrent Versions System (CVS) which uses the diff algorithm to show the differences between different revisions of a given program.

However, these two solutions cannot be generalized as they do not understand the all structure of the data. Later, structured document differencing algorithms were proposed to fit the requirement of structured data for Latex and nested-object documents such as LaDiff [11] and MH-Diff [12] algorithm. Algorithms are proposed also for XML documents such as diffX [13]. It is based on the bottom-up mapping and the DOM-hash to reduce the size of the trees to compare. Our change detection algorithm is inspired from the diffX [13] algorithm, first because it proved its worth in change detection compared to the other solutions related to structured docu-ments. Second, it is based on the tree structure considering a structure similar to the Hierarchical Hash Tree.

C. Network protocols for file synchronization

Various networking protocols and architectures are used to ensure data synchronization including both the notification and the data transfer. In fact, a part of the infrastructure sends messages to the concerned connected clients. It notifies them that changes have occurred and the data transfer for synchronization should start. For example, Dropbox [9] uses the HTTP for the data transfer and the long HTTP polling for the notification. Regarding Google Chrome synchronization, the data transfer is also based on HTTP and the notification exploits an existing XMPP-based Google Talk server. REST API is also used to ensure data replication in multiple solutions such as

CouchDB [15]. Indeed, RESTful applications are mainly based on HTTP protocol.

Choosing the best protocol for the data exchange is very important to the synchronization protocol efficiency. In fact, it is crucial to reduce the amount and the size of messages exchanged between the both end points. In this paper, we address the introduction of the WebSocket API and protocol in the architecture of synchronization.

III. OVERALL SYNCDS ARCHITECTURE

The goal of our framework is to ensure the data synchronization between a Local Digital Safe and a Cloud Digital Safe while considering the probative value. As a first step, we need to define the architecture as well as the messages exchanged between its different entities. The architecture is divided into

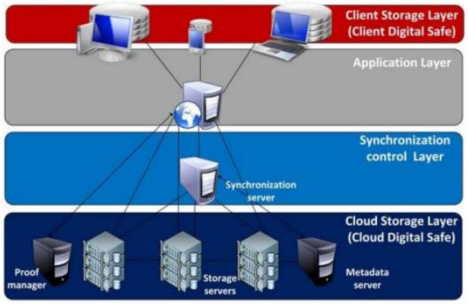

Fig. 1. Digital Safe synchronization Architecture

- Client Storage Layer: The novelty in the architecture is the non-proprietary characteristic. The data are stored securely in a Local Digital Safe. This Safe is based on the HTML5 Local Storage API with additional security considerations. In fact, we add into the existent APIs the confidentiality by encrypting the data locally, the data integrity and metadata integrity. These security enhancements are the subject of previous work, and more details can be found in [16].

-Application Layer: This part includes the web application with the different used APIs. We introduce two main modules. The first one, Digital Safe, implements the AFNOR specifica¬tions to manage locally the data stored using the File System API. The second module, synchronization, is used to detect the user's operations applied on the Local Digital Safe and to record them. It manages then the exchanged messages between the Local Digital Safe and the Cloud Digital Safe following the SyncDS protocol.

-Synchronization Control Layer : To go beyond the commercialized solutions and to have the best performances, we choose the WebSocket protocol to ensure the bidirectional communication between the local and the remote Digital Safe. The synchronization management server handles the different synchronization requests and responses as well as the conflict resolution. It also notifies the devices concerned by the modification to propagate the changes. Our proposed algorithm, based on HHT, is introduced at this level.

- Server Storage Layer: As introduced in [1], the Cloud Digital Safe is a standardized architecture that provides a secure environment for storing sensitive document. This en¬vironment fully fits both the user requirements and Cloud security challenges. This Cloud Digital Safe is composed of three main components: Metadata server, storage servers and Proof Manager.

IV. SYNCDS SYNCHRONIZATION PROTOCOLS

A. Overview on synchronization Steps

After the architecture definition, we need to itemize the syn-chronization protocol. SyncDS ensures data synchronization in a standardized Digital Safe context.

Our protocol holds three phases as depicted in the figure 3:

-Offline phase: when the user goes offline, the Client Digital Safe stores in a log file the different operations performed locally. This strategy is the unique which can be used even if it needs storage capacity. In fact, it guarantees the non¬proprietary characteristic and avoid the use of solutions which depend on the used operating system. These operations are detected through the enhanced HTML5 fileSystem APIs spec¬ification and the Application cache API. They are stored then using the HTML5 WebStorage API.

-On connection phase: It is a two way synchronization that includes two steps. In the first step, the client posts changes performed when it was offline. This step is based on the operation approach. In fact, the log file is sent to the server. The server then applies on his version the changes as listed in the log file.

In the second step, the server reveals changes performed on his side when the user was offline. These changes can be performed by the same user in a different device or by another user who shares a part of the file system. In this step, the differencing approach is adopted. In fact, as depicted in the figure 3, the Client Safe sends an abstract of its file system (1) to the server. The server then compares the received abstract with his version, detects changes (2) to send them back to the user. We propose a new algorithm based on HHT to detect the changes. This algorithm will be detailed later in section IV-B. In case of multiple devices connected at the same time, the problem of conflict resolution is raised and many techniques are already proposed to solve it [7] [5].

- Online phase: It is a two way synchronization. Changes made on the client side are sent to the server and changes, made by different devices and synchronized to the server, are sent to the user. This step is based on the operation approach. The novelty and originality of our protocol are the introduction of the WebSocket to send data from the

client to the server. It is also adopted by the server to send notifications and data to the client.

B. Changes detection during the on connection mode

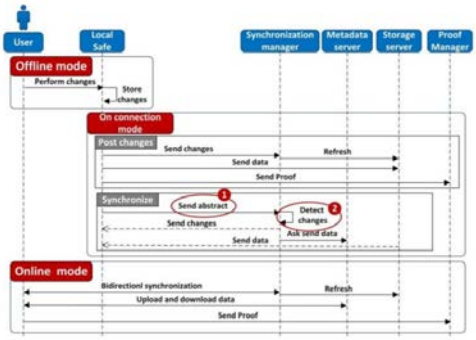

Fig. 2. Overview on the synchronization protocol

1) Abstract structure: Among the metadata information, we extract the ID, the path, the type and the hash of the object (file or directory) to build the abstract. The abstract, in this context, can be modeled into a rooted and ordered tree: rooted as it has a root directory and ordered as there is a hierarchical relation between files and directories. The special feature of the abstract, in our proposal, is the introduction of the Hierarchical Hash Tree (HHT). In fact, two main functions can be considered when dealing with the HHT.

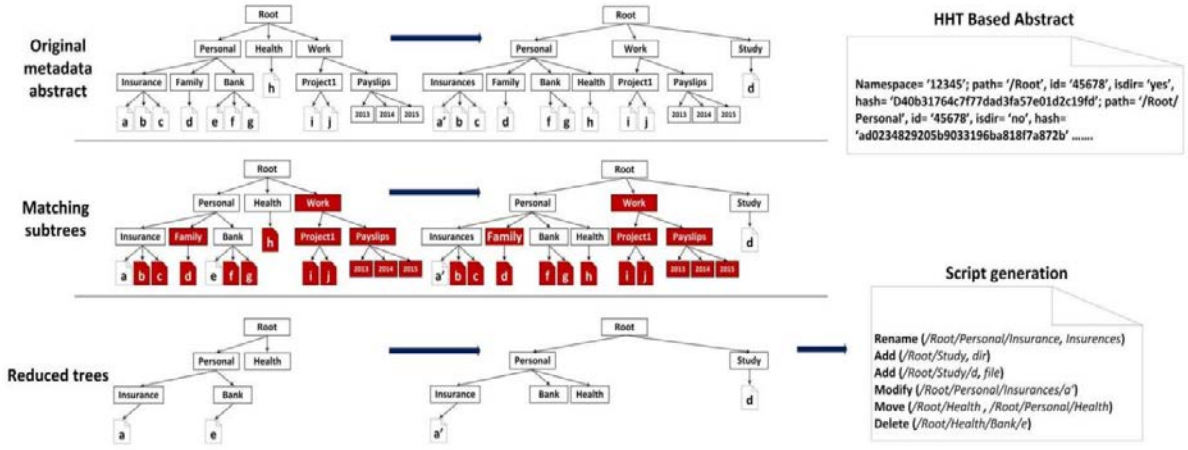

Fig.3 Changes detection algorithm.

first to concatenate the different hashes of the directory's objects (files and directories). Then, we compute the hash of the concatenation's result. The concatenation should follow a specific order of objects to have a unique hash. In our case, we order the objects in the ascending number of their IDs.

$$Hash_D(dir) = Hash_{MD5}(Hassh_F(f) + Hash_D(d))$$

2) Changes detection algorithms: The whole algorithm is composed of two algorithms: isolated subtree matching and edit script generation.

a) Isolated subtree matching (Algorithm 1):

While comparing both versions of abstracts, the function match(x,y) where x H T1 and y H T2 detects the subtrees that has not bee^ changed in the trees T1 and T2:

$$match(x,y) = \begin{cases} true & \text{if } Hash_{F/D}(x) = Hash_{F/D}(y) \\ \{ & Name(Parent(x)) = \\ & Name(Parent(y)) \\ false & \text{otherwise.} \end{cases}$$

Algorithm 1 Isolated subtree matching

```
1: procedure DETECT MATCHING(T 1, T 2)
2: Input T1, T2: tree
3: Output T 1, T 2: tree
4: Traverse T1 in a level order and top-down
5: let x be the current node
6:     if  y  T2/ match(x,y)=true  then
7:         Delete (subtree(x), T1);
8:         Delete (subtree(y), T2);
9:         if Name(x) = Name(y) then
10:            Rename(x,Name(y))
11:        end if
12:        if  y  T2/ Hash(x)=hash(y) and Par-
   ent(x)=Parent(y) then
13:        Mark (y, x);
14:
15:        end if
16:    end if

17: End-traverse
18:    if  y  T2/ Mark(y)=True then
19:        Copy (x, Name(y) );
20:        Delete (subtree(y), T2);
21:    end if
22:
23: end procedure
```

Using a top-down tree analyses, the first algorithm looks for the set of subtree that matches between both trees, i.e., matching the tree T1 and T2 leads to find the pair(x,y) where the subtree of T1 rooted at x and the subtree of T2 rooted at y match (using equation (3)).

While a match is found both subtrees are erased from the trees. In case of unmatched, the algorithm recursively passes to the level bellow of the tree T1 and continues the detection of matched subtree. The outputs of this algorithm are two reduced trees.

b) Edit script generation (Algorithm 2): :

The edit script contains the different operations which applied on the tree T1, we obtain then the tree T2. The algorithm in this phase follows a down-top traversal of the new version of the trees T1 and T2. The order of operation detection in this algorithm is important and is considered as follows: rename, copy, move, add ,modify, delete. When the changes have been detected for an object of the tree, this object will be deleted from the reduced tree.

In this algorithm, the different used function are detailed bellow:

Type(n): returns the type of the node n. It can be a file or directory;

```
Algorithm 2 Script generation
 1: procedure GENERATE SCRIPT(T1, T2, S)
 2: Input T1, T2: tree
 3: Output S:List
 4: Traverse T2 in a level-order sequence and down-top
 5: let y be the current node
 6:     while ∃ x ∈ T1 / Match(x,y)=True do
 7:         if ∃ x ∈ T1 / Match(x,Parent(y))=True then
 8:             y==Parent(y) ;
 9:         else break;
10:         end if
11:     end while
12: Add(Path(Parent(y))/Path(y), Type(y))
13:     if ∃ x ∈ T1 /( Name(y) = Name(x) and Type(y) =
    Type(x)) then
14:         Modify (Path(x));
15:     else if ∃ x ∈ T1 / Parent(y) = Parent(x) and Name(y)=
    Name(x) and Type(y) = Type(x) and Hash(x) = Hash(y)
    and parent(y) ∈ T1 then
16:         Move (Path(x), Path(Parent(y)));
17:     else if ∃ x ∈ T1 / Parent(y) = Parent(x) and Name(y)
    = Name(x) and Type(y) = Type(x) and Hash(x) = Hash(y)
    then
18:         Rename(x,Name(y));
19:     end if
20: End-traverse
21: Traverse T1 in a level-order sequence and down-top
22: let x be the current node
23:     for x ∈ T1 do
24:         if ∄ y ∈ T2 / Hash(x)= Hash(y) or
    Name(x)=Name(y) then
25:             Delete(x)
26:         end if
27:     end for
28: end procedure
```

Path(n): returns the path of the node n;

Name(n): returns the name of the node n;

Subtree(n): returns the subtree rooted at the node a and extracted from the Tree T. This subtree matches to the repository n with its content;

Match(ni,n2):verifies if two nodes n1 and n2 from two different trees are identical;

Parent(n): returns the node parent of the node n;

HashF/D(n): returns the hash of the file or directory; Mark(n): puts a mark on the node n to detect at the end of the first algorithm the copy operation.

V. IMPLEMENTATION AND PROOF OF CONCEPT

As a proof of concept and a proof of the protocol efficiency, we focus mainly on three parts of the synchronization archi¬tecture.

- The Chromium browser: We focus, in this paper, on the Filesystem API. We add the encryption of files content in the

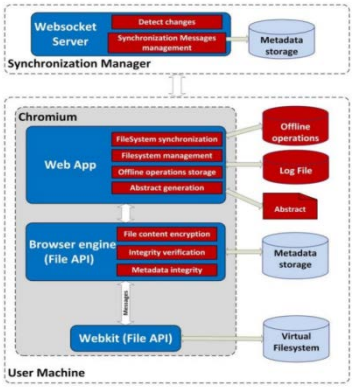

Fig. 4 Implementation of the synchronization architecture and protocols.

client side. We add also the data integrity with the verification of the file hash and the metadata integrity by encrypting the file name when stored in the indexed database by the chromium browser.

- Synchronization in the Web application: A HTML5 web application is developed based on the enhanced File System, the basic Webstorage, the Application Cache and WebSocket APIs. At this level, the abstract of the file system is build following the HHT structure. This application allows the user to manage his Digital Safe, store the operations when he is offline and to ensure the synchronization of his Digital Safe content following the SyncDS protocol.

- Synchronization in the WebSocket Server: To introduce the WebSocket server, we use the web server Apache with its extension mod_pywebsocket. We add in the server side the detection of changes by comparing two abstracts (the one sent by the Client Digital Safe and the other extracted from the Cloud Digital Safe). The comparison of both abstracts is implemented using the Java language and more specifically using the TreeModel interface.

VI. ANALYSIS OF THE SYNCDS PROTOCOL

A. Efficiency perspective

SyncDS is a file synchronization protocol that focuses on guaranteeing a high quality and minimal resource consumption within a secure environment. It presents two main novelties. The first one is the non-proprietary characteristic of the synchronization architecture which has a great importance to integrate a broad range of competing products and devices. In fact, we enhance HTML5 Local Storage APIs to store locally

user's data in a client Digital Safe. We use also the HTML5 WebSocket API for the data transfer between the Cloud and client Digital Safe.

Property 1: The low computation of the data and metadata at the client side is respected within our algorithm. If we compare SyncDS with the already existent synchronization protocols, the unique additional computation at the source machine is the extraction of the Hierarchical Hash Tree. We are keen on generating the abstract only once during the on connection phase.

Property 2: Introducing the Hierarchical Hash Tree im¬proves the efficiency of the change detection between the file systems of the client and the server. In fact, processing reduced version of trees is more efficient than processing the whole trees. This minimizes systematically the number of operation to generate the script.

To highlight the performances of HHT integration, we compare, in figure 5, the time needed for the script generation with and without the integration of the HHT into the abstract structure. In fact, comparing these two cases is equivalent to comparing the points of differences between our proposed architecture and the already commercialized one under the same experimental conditions. .

. The results show that the change detection in the basic protocol

Property 3: The number of synchronized files is reduced while using the Hierarchical Hash Tree. For classic changes detection [10] [17], they rely on objects name to detect the changes. Therefore, when an object is renamed, copied or moved, the whole object with its content is synchronized since it is considered as a new element in the new version of the file system. With HHT, and as the hash is introduced into the conditions of detection, these repositories and files are no more considered as new elements and will not be synchronized as their content already exists in the storage servers.

Property 4: Adopting the WebSocket protocol reduces the messages exchanged between the client and the server. This reduction is justified by the bidirectional communication ensured by WebSocket as the server can send any messages to the client without needing a request from the client. Even the size of messages is reduced using the WebSocket protocol.

We compare in this part our protocol with the existent synchronization solutions which use the HTTP as the basic protocol of com- munication between different entities of the architecture. As shown by the results, the WebSocket protocol is by far the most efficient, especially with large files.

B. Security perspective

The architecture of synchronization is based on the Digital Safe context. This context focuses mainly on adding the probative value and preserving the integrity of a digital object over the time.

Introducing the Hierarchical Hash Tree into the synchro¬nization participates directly to ensure the security of data in addition to the performances enhancement. In fact, verifying the hash value of the root directory of one file system detects systematically if one object of the file system has been altered by a third party, or the file system keeps its original version.

In this case, it is no longer necessary to check the hash of

objects one by one to verify the integrity.

VII. CONCLUSION

In front of the various owned devices and the need of synchronizing the data between them, ensuring an efficient file synchronization protocol is crucial. In this paper, we propose an architecture and a protocol that ensure file synchronization in a probative value Cloud. Two keynote novelties of the SyncDS protocol are highlighted: first, the integration of the Hierarchical Hash Tree into the metadata abstract and second, the non-proprietary characteristics with the adoption of HTML5 APIs. Our experimental results show that using the new proposed framework, reduces the time of change detection and therefore, reduces the time of file synchronization across devices .

The conflict resolution is raised in our architecture in case of multiple connections at the same time. As future work, we will deal with the interference of the conflict resolution strategies with the execution of our protocol.

REFERENCES

[1] M. Msahli and A. Serhrouchni, "Sbaas: Safe box as a service," in 9th IEEE International Conference on Collaborative Computing: Network- ing, Applications and Worksharing, Nov 2013.

[2] "Afnor groups." [Online]. Available: http://www.afnor.org/en

[3] M. D. N. Jain and R. Tewari., "Taper: Tiered approach for eliminating redundancy in replica synchronization," in . In Proc. of the USENIX Conference on File And Storage Systems, 2005.

[4] S. Agarwal, D. Starobinski, and A. Trachtenberg, "On the scalability of data synchronization protocols for pdas and mobile devices," IEEE Network, Jul 2002.

[5] B. Xianqiang, X. Nong, S. Weisong, L. Fang, M. Huajian, and Z. Hang,"Syncviews: Toward consistent user views in cloud-based file synchro- nization services," in Sixth AnnualChinagrid Conference (ChinaGrid),, Aug 2011.

[6] H. Yan, U. Irmak, and T. Suel, "Algorithms for low-latency remote file synchronization," in The 27th Conference on Computer Communica-tions. IEEE INFOCOM 2008, April 2008.

[7] C. Liang, L. Hu, Z. Lei, and J. Wang, "Synccs: A cloud storage based file synchronization approach," Jul 2014.
[8] Benjamin, C.Pierce, and J. Vouillon, "What's in unison? a formal specification and reference implementation of a file synchronizer," in Tech. rep. MS-CIS-03-36, Department of Computer and Information Science, University of Pennsylvania, 2004.
[9] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: Understanding personal cloud storage services," in
[10] Proceedings of the 2012 ACM Conference on Internet Measurement Conference, ser. IMC '12, 2012.
[11] A. Tridgell and P. Mackerras, "The rsync algorithm. technical report tr- cs- 96-05, department of computer science," in The Australian National University, Canberra, Australia, 1996.
[12] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change detection in hierarchically structured information," in Pro-ceedings of the 1996 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '96, 1996.
[13] S. S. Chawathe and H. Garcia-Molina, "Meaningful change detection in structured data," in Proceedings of the 1997 ACM SIGMOD Interna-tional Conference on Management of Data, ser. SIGMOD '97, 1997.
[14] R. Al-Ekram, A. Adma, and O. Baysal, "diffx: An algorithm to de¬tect changes in multi-version xml documents," in Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research, ser. CASCON '05, 2005.
[15] N. Jain, M. Dahlin, and R. Tewari, "Taper: Tiered approach for eliminat¬ing redundancy in replica synchronization," in In Proc. of the USENIX Conference on File And Storage Systems, 2005.
[16] J. C. Anderson, J. Lehnardt, and N. Slater, "Couchdb the definitive guide." [Online]. Available: http://guide.couchdb.org/
[17] M. jemel and A. serhrouchni, "Security assurance of local data stored by HTML5 web application", in the 10th international conference on information assurance and security, Okinawa, Japan,