Combining Mobile and Cloud Storage for Providing Ubiquitous Data Access*

João Soares and Nuno Preguiça

CITI/DI-FCT-Univ. Nova de Lisboa
Quinta da Torre, Portugal

Abstract. Users increasingly own, and use, multiple computing devices. To be able to access their personal data, at any time and in any device, users usually need to create replicas in each device. Managing these multiple replicas becomes an important issue.

In this paper we present the FEW Phone File System, a data management system that combines mobile and cloud storage for providing *ubiquitous* data access. To this end, our system takes advantage of the characteristics of mobile phones for storing a replica of a user's personal data, thus allowing these devices to be used as *personal and portable file servers*. As users tend to always carry their mobile phone with them at all times, these replicas are the basis for providing high data availability, and keeping replicas *automatically* synchronized.

Our system also uses other replicas located in web servers and cloud storage systems, to reduce the volume of data stored, and transferred to/from mobile phones, by maintaining only the information needed to obtain them.

1 Introduction

Users increasingly own, and use, multiple computing devices, from desktop computers, to laptops, tablets, consoles and mobile phones. In such an environment, data availability is an important issue, as users want to access their personal data everywhere, independently of their current machine or location.

To address this problem, users tend to rely on at least one of the following available solutions: *i*) on-line storage services (e.g. Dropbox [1], Google Docs [3]), and/or *ii*) portable storage devices (e.g. USB flash drives). While both solutions aim at providing "ubiquitous" storage space, they also force users to deal with additional problems. On-line storage services force users to trust third parties for storing their personal data. Both solutions force users to maintain synchronized replicas of the stored data, for minimizing losses due to possible device failure or for guaranteeing access in case of network disconnection. In many solutions (e.g. Google Docs, USB flash drives), synchronization is done manually, and only strict discipline avoids replicas from diverging.

* This work was partially supported by CITI and FCT/MCTES project # POSC/EIA/59064/2004, with Feder funding. João Soares was partially supported by CITI and FCT/MCTES research grant # SFRH/ BD/ 62306/ 2009.

E. Jeannot, R. Namyst, and J. Roman (Eds.): Euro-Par 2011, LNCS 6852, Part I, pp. 516–527, 2011.

In this paper, we present the FEW Phone File System (FEW), a data management system for providing high data availability to mobile users. While sharing some of the goals and solutions with existing systems [19,9,12,2,16,15], FEW provides a unique solution for offering users *ubiquitous* access to their personal data, independently of their location, device and/or network connectivity. To this end, FEW builds on the characteristics of current mobile phones, taking advantage of their *i*) storage capacity; *ii*) wireless communication capabilities (Wi-Fi and/or Bluetooth); *iii*) mobility of such devices.

In FEW, the mobile phone of a user acts as his *personal and portable file server*, storing and maintaining replicas of his personal files. This allows our system to provide ubiquitous access to these files, since users tend to carry their mobile phones with them at all times. The Wi-Fi and Bluetooth communication capabilities guarantee access to these replicas from any computer, in a location independent manner, without the hurdles of needing extra cables. The system includes an optimistic replication solution, allowing for replicas to be created and accessed when needed, including a *novel scalable update tracking solution*, and a *reconciliation algorithm based on commutative operations*.

As a large number of users' files are currently stored, or are obtained from remote web sources, FEW includes a data source verification mechanism to record the alternative sources for each file. This allows the integration of online storage systems as alternative data sources for each file, achieving both the reduction of communications, as well as data that needs to be stored in the mobile phone, while still providing high data availability. Additionally, FEW includes a data transcoding mechanism for minimizing storage consumption due to multimedia data that needs to be stored in the mobile phone - these two mechanisms allow, for example, the mobile phone to store only thumbnails of photo collections, while keeping information on how to obtain full fidelity photos from on-line systems, like Flickr, Facebook, etc.

The remainder of this paper is organized as follows: Section 2 presents the design of our systems; Section 3 details the advanced mechanisms to address limitations of using mobile phones; Section 4 presents an evaluation of the system; Section 5 discusses related work; and Section 6 concludes the paper with some final remarks.

2 System Design

The FEW Phone File System (FEW) is a data management system designed to provide high data availability to users, allowing them permanent access to their personal data, across multiple computers, independently of location, ownership, and network connectivity. In this section we present the system general design and synchronization algorithms.

2.1 Architecture

FEW is based, primarily, on a client-server architecture, where mobile phones act as *personal, and portable, file servers*, storing replicas of the personal data

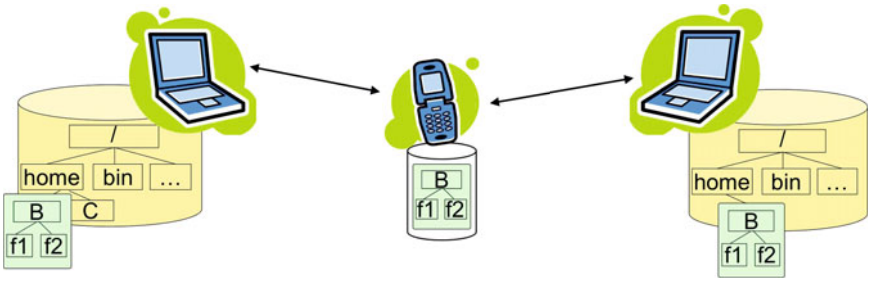

Fig. 1. FEW Phone File System architecture

of some user. A typical FEW configuration includes a set of computing devices, called nodes, accessed by a common user, and, at least, one mobile phone, as depicted in Figure 1. In the remainder of this paper, a mobile phone acting as a file server will be addressed as a *mobile server*.

Each node is responsible for managing its replicas of the user's files, which are organized into collections named *containers*. Containers are collections of files stored under the same common path name, and can be seen as subtrees of a file system, similar to volumes in Coda [18]. FEW allows containers to be created, in any client node, from any existing directory in the file system, or when replicating an existing container on a new node.

Applications access these files as any other files, i.e., using the file system interface. This allows FEW to provide data availability, without forcing applications to be modified. To this end, clients intercept and handle all file system calls executed on the files managed by the system. In the current prototype, we rely on Fuse [6] (and Fuse-J java-bindings) for implementing the client side. Typically, mobile phones only include the server component, while computer nodes run both the client and the server component of the system. This allows computers to synchronize files with mobile servers, using a client-server model, and also with other active client nodes, in a peer-to-peer fashion, as described later.

FEW fully replicates a container's complete namespace in every replica, thus allowing users to have a coherent view of its contents in every node. File contents, on the other hand, do not need to be fully replicated. As explained later, the system includes several mechanisms that allow the system to use partial or full replication policies. If necessary, file contents can be fetched on-demand when applications access files not locally replicated.

For improved performance, we use optimistic replication techniques [5] to allow replicas of the files to be created in any client computer that accesses them[1]. This way, files can be modified in any computer without requiring immediate server communication, thus addressing performance and energy limitations of mobile phones. Data consistency is maintained by a periodic synchronization

[1] Users can decide in which computer these replicas can be created, and in which replicas can be long-lived, thus addressing privacy concerns.

process, used for propagating updates from clients to servers, and vice-versa. This process includes a reconciliation mechanism for resolving possible conflicts due to concurrent updates.

2.2 Synchronization Process

To preserve consistency, allowing users to always access the latest version of their files, FEW relies on a periodic synchronization process for propagating updates between replicas of a container. Replicas are synchronized using an epidemic approach, by establishing pair-wise communication sessions. Although any node can synchronize with any other node, mobile servers tend to function as mediators for propagating updates between replicas.

This approach has several benefits. First, it allows our system to provide eventual consistency, i.e., over time, an update executed in a node will be propagated to all other nodes, even to nodes that have no network connectivity (besides short range connectivity to the mobile phone). Second, since users are expected to always carry their mobile phones with them at all times, maintaining the most recent version of a container on a mobile server allows FEW to always provide users access to the most recent version of their data.

A synchronization session is automatically initiated whenever a node detects a nearby mobile server. During this session, local updates, i.e., updates performed on a computer node, are propagated to the mobile server, while missed updates are propagated from the server to that node. This process is re-executed periodically while a mobile server is nearby. As a result, we expect a mobile server to always store the most recent version of its containers.

The synchronization process also allows nodes to gain knowledge of other existing replicas. Whenever a container is replicated on a node, the node's address is stored on the server, thus creating additional sources for obtaining the data. This information is exchanged during the synchronization process.

This allows the basic synchronization process to be extended using node-to-node synchronization. This way replicas are kept synchronized, even if the mobile server cannot be used to efficiently propagate updates among them. In this case, a node uses this information to form an overlay network for exchanging updates with existing replicas. Two situations for using such an approach have been identified. First, when the user forgets his mobile phone, it is obvious that synchronization must be performed using peer-to-peer communication. Second, when files are too large it might be too costly to store them in the mobile phone.

Synchronization Algorithm. FEW uses a two stage synchronization algorithm. First stage synchronizes the name space, i.e., it propagates unknown directory updates, while the second stage synchronizes data, i.e., files contents.

Object identifiers. Internally, all objects handled by FEW are addressed by unique identifiers (UID), assigned to each object (file or directory) when created, or during the container creation process, remaining the same during the

lifetime of that object, independently of name changes. Other information (i.e. meta-data) is also maintained for each object in a container. This information includes common file system attributes, such as file names, access permissions, etc., and system-specific attributes, such as information for tracking dependencies among versions, a digest of its contents, a list of alternative file sources, etc. This information is essential for the synchronization process.

Name Space Synchronization. For synchronizing the name space of a container, FEW maintains a log with name space updates, and an associated version vector [10]. Each log entry includes all the necessary information for reproducing the operation in other nodes, i.e., a time stamp of the operation, the type of operation (create, delete, rename), the UID of the updated object, its version, and the UID of the parent directory, while the version vector allows for recording the number of updates performed on each replica of the container. During synchronization, each node exchanges version vectors, thus allowing missed updates to be determined. The respective log entries are then exchanged, and reproduced.

Concurrent updates are deterministically resolved using the principles of commutative replicated data types [14]. Concurrent operations have been designed in a way that a deterministic result is obtained independent of their execution order, thus guaranteeing that all replicas converge to the same state.

Data Synchronization. For tracking data dependencies, FEW uses a novel approach. Updated files are initially signaled using an *'updated'* flag associated with the file and all parent directories. During synchronization, each file marked as updated is synchronized with the peer and only at this moment the version vectors are modified. Unlike the traditional approach, if only one replica has been modified and only one peer has an entry in the version vector, it is that entry that is updated to record the new version, independently of the replica in which the file has been changed. This approach allows the reduction of the number of entries in version vectors. In our scenario, where we expect a node to synchronize with a single (or a small number of) mobile servers, it is possible to keep the number of entries in version vectors equal to the number of mobile servers, as the updates performed in a node will always be reflected in the server's entry.

Additionally, in FEW, version vectors of directories summarize the updates in the subtree. Thus, during synchronization it is only necessary to propagate the version vector of the root to be able to determine which files need to be synchronized. This allows the volume of exchanged data, during synchronization, to be proportional to the number of updated files, rather than the total number of files in a container. Concurrent updates to the same file are automatically resolved by deterministically selecting a predominant file version. Additional details of the synchronization process can be found in [8,7].

3 Advanced Mechanism for Using Mobile Phones

Relying on mobile phones as file servers, imposes the following limitations that need to be addressed: *i*) limited communication bandwidth, leading to higher

data access times, compared to local hard drives; *ii*) insufficient storage capacity; *iii*) limited energy resources; and *iv*) low reliability of mobile phones [17]. Next we briefly present some of the solutions used to address these problems.

3.1 Minimizing Communications

As in any distributed file system, accessing all files directly on the file server proves impractical [11,18]. This results from limited bandwidth and high latency limitations, which increase access times, when compared to local hard drives. Also, since in our case the file server is a mobile phone, server communications are highly energy consuming operations, thus minimizing them in essential.

As explained before, FEW addresses this problem by creating temporary or long-lived replicas in the clients nodes. The system relies on periodic synchronization to keep replicas up-to-date, exchanging, during these interactions, information that is proportional to the number of updated replicas in the system (as in Cimbiosys [15]). This approach helps minimizing communications with the mobile phone, for both file access and synchronization, providing faster access times to the users files, improving the user experience, while also minimizing power consumption of due to communications.

3.2 Improving Storage Usage

Although mobile phones have increasingly larger amounts of storage capacity, they offer reduced storage space when compared to the current capacity of hard disks, and the amount of personal data users tend to store. This way, we expect to be impossible storing replicas of all files kept by an user. FEW addresses this limitation in different ways.

Container Set. Since FEW is designed to manage a set of containers, we believe users would assign, to different containers, different types of data. For instances, one container would have personal documents, like word documents, text files, spread sheets, etc., while others would contain video files, audio files, etc.. This way users can select which containers should always be available, i.e., replicated on the mobile server, and which are less important. This can be manually adjusted, thus allowing users to always keep relevant data "close by".

Integrating Remote Storage Systems. Users tend to store significant number of resources obtained from remote sites, or that have additional remote copies. Among this data, we can include, not only, files downloaded from Web sites, for example, *'pdf'* files, but also data uploaded by the user to Internet workspaces, or other remote storage systems, such as cloud storage services. Since these files tend to be preserved for long periods of time, and, in some cases, remain unchanged during this time, there is no reason why these remote sites cannot be used as alternative data sources for obtaining data.

With this in mind, FEW includes a Data Source Verification (DSV) mechanism that allows it to automatically keep track of additional sources for the files

managed by the system. DSV is a modular, plugin based component, allowing different services to be used as alternative data sources, simply by choosing the corresponding plugin, including Web servers, CVS servers, and Cloud storage systems. For instance, the current HTTP generic plugin acts as a proxy, monitoring HTTP connections. For each HTTP operation, it computes the digest for the obtained result, storing it, as well as the associated URL, for a short period of time.

When a file is created or updated in a container, the system checks with the DSV if the new contents were remotely obtained (by comparing its digest). If those contents were obtained remotely, the source URL is added, as meta-data, to the corresponding file. This information is used during the synchronization process, allowing our system to retrieve remotely stored files directly from these sources. This way, FEW prevents those contents from being stored on the mobile server, storing only their meta-data. FEW can still provide high data availability for these files, by leveraging on the alternative data sources associated with those contents for retrieving them. Doing so allows our system to reduce storage requirements.

Some DSV plug-ins require more complex solutions the one used in the HTTP plugin. For example, a CVS plugin requires additional information to work correctly, since remote files have an associated version. Thus, the CVS plugin checks, not only if a file's pathname is under control of a CVS server, but also if the version of the remote file is identical to the one stored locally. If so, the CVS server can be used as an alternative source for retrieving the file. This information is added as meta-data, thus allowing our system to retrieve these contents during synchronization.

Currently we are developing a plugin for integrating FEW with Cloud storage services, such as Dropbox [1]. As in the CVS plugin, this plugin requires additional information from the user, such as login credentials. With this information, the plugin can use the Dropbox API to create an authenticated session, using it for browsing the contents remotely stored.

Maintaining Consistency of Remote Replicas. Our system also allows the automatic update of remote replicas whenever a local replica is updated. To this end, the DSV plugin includes methods to allow these updates to be performed. For example, the HTTP plugin relies on using HTTP POST and PUT operations.

During synchronization, if remote replicas cannot be updated, either because updates are not supported by some plugin or were unsuccessful, the mobile phone is used to store the most recent version of the file. Our experience says, that most resources downloaded from Web pages are kept unchanged by users. Only personal data is regularly updated, and this information is normally stored on services that allow remote update operations to be performed.

Multimedia data. Another mechanism designed to reduce storage requirements relates to multimedia files. With the proliferation of digital cameras, and other multimedia devices, users tend to store significant amounts of multimedia contents. Since these files tend to consume considerable amounts of storage space,

it is unreasonable to store them on a mobile phone. Although some of these files can be addressed using the previous mechanism, others can not.

For these files, FEW relies on data transcoding techniques [13]. These techniques allow for multimedia contents to be adapted accordingly to the available resources on mobile servers, storing only lower fidelity versions of multimedia files. FEW automatically performs data transcoding during synchronization, immediately before transferring multimedia data to the mobile server, allowing the level of fidelity to be specified for each device.

The Data Transcoding (DTC) module includes a set of plug-ins for transcoding different types of data. In our current prototype, we include support only for transcoding a very limited number of multimedia formats, using existing applications. We also include support for transcoding generic files to an empty file. This provides support for implementing partial replication of a container. Additionally, users can configure the transcoding parameters for each supported file type. allowing these parameters to be changed according to the available storage space. For example, users can define a "more aggressive" level of transcoding when the available storage space is low.

Combining this mechanism with the previously described DSV mechanism, allows lower-fidelity versions to be used only when they are satisfactory, or when no connectivity to an alternative source is available, since the full fidelity versions can be downloaded from the alternative sources, whenever needed.

4 Evaluation

In this section we present an evaluation of the FEW Phone File System, focusing on the advanced modules: DSV and DTC. We evaluate both the importance of the modules and their performance.

4.1 Importance of DSV and DTC

For evaluating the importance of DSV and DTC modules, we have studied the percentage of user files obtained from remote web sources, which can be handled by DSV; and the percentage and relative volume of multimedia files, which can be addressed by DTC.

As we could not run our system for a long enough period and with a large enough number of users to obtain relevant statistics, we have obtained statistics analyzing the personal data of 5 users. For determining which files had been obtained from a remote source, we have used the application-specific attributes with the URL added by the Safari web browser in Mac OS X. Obviously, this is a lower-bound estimation of the files obtained remotely, as some of these users reported that they also use other browsers that do not add this attribute. Additionally, files obtained using other programs are also not computed.

Table 1 shows the type and the *relative volume* of data stored by users (office data includes, not only word documents, spreadsheets and presentations, but also digital documents such as "pdf" files). As we expected, users tend to store

Table 1. Statistics on personal files

Data Type	Relative Volume	Data Source	
		Locally Created	Web Transferred
Multimedia	56%	98%	2%
Archives	17%	68%	32%
Office	6%	53%	47%
Sources	0%	95%	5%
Other	21%	79%	21%

considerable amounts of multimedia data. In average we found that more than 50% of the files stored by current users are multimedia data. Since these files tend to be large, the use of the Data Transcoding module is essential for allowing FEW to achieve its goals.

Table 1 also presents the data obtained from the Internet using Safari. From these results, it is possible to observe that approximately 14% of the personal data has been obtained from remote sites. The largest amount of downloaded files are office and archive files, which results from the fact that users tend to store large numbers of digital documents, such as "pdf" files, and that archives are largely used to enclose other resources, such as source files. The low percentage for multimedia files can be justified by the fact that most of these files are: (1) copies of data users own(e.g. CDs) or have created (e.g. photos from cameras); (2) obtained using applications other than browsers - e.g. iTunes and peer-to-peer applications. For source files, we know that some users have a large percentage of their files in version control systems, but we could not quantify the percentage. For some multimedia files - e.g. photos, it is common for users to store them in remote sites. Thus, the files that could be handled by DSV is expected to be much higher.

These results show that keeping track of alternative sources can reduce the need for storing the contents of files, thus showing that the DSV module can be a good solution for minimizing the data that needs to be stored on a mobile server. Additionally, as multimedia files seem to be the category that has less additional sources, DTC seems a good complement to DSV.

4.2 Performance Impact of the DSV and DTC

In this section we present performance results obtained when synchronizing nodes using the DSV and the DTC modules. The results were obtained using our Java/Android prototype. The mobile server runs on an HTC Magic mobile phone running Android 1.5, while the client nodes are desktop computers with an Intel Core 2 Duo T8300 @ 2.4 GHz process, running Linux Ubuntu 9.10. Devices communicate using a Wi-fi network.

To determine the impact of the DSV module, we measured the time for synchronizing a new node with a mobile server, and those obtained performing the same operation using a third node as an additional source for transferring data. The results, presented in Table 2, show performance gains when using the DSV

Table 2. Synchronization times with DSV

Num. files	Total Size	Sync. time w/ DSV	Sync. time wo/ DSV
83	450 KB	5.984s	6.502s
97	2 MB	7.806s	9.171s
357	5 MB	25.467s	35.689s

Table 3. Synchronization times with DTC

Num. files	Total Size	Sync. time w/ DTC	Sync. time wo/ DTC
1	9 MB	5.211s	8.775s
2	8 MB	9.216s	18.556s

modules, even for small data volumes. The actual impact on performance is directly related with the bandwidth and latency of the connection with the server. Besides the performance improvement, this module allows for a considerable reduction of the volume of data transferred from the mobile phone to the node, also reducing power consumption of the mobile device.

For evaluating the impact of the DTC module, we measured the time for synchronizing a container with one and two 14 mega-pixels (4672x3104 pixels) digital photos from a node to a mobile server. Table 3 compares the values when propagating the full fidelity photos and when using the DTC module (the two photos were transcoded to a resolution of 1024x680 pixels, and color depth from 32 to 24 bits per pixel, for a size reduction from 9 MBytes to 63 kBytes). Results show a significant performance improvement and, above all, present a significant reduction in the volume of data that is transferred to and from the server. Combining these two modules allows users to access their full-fidelity data, while reducing the volume of data transferred and stored on the mobile phone, thus improving performance while reducing power consumption.

5 Related Work

Some distributed file systems (e.g. Ficus [4], Coda [18]) include support for mobile computing environments. However, the complexity associated with setting up a new server and using it in a network with private networks and firewalls lead most users to prefer using portable storage devices to transport their data.

Other solutions for addressing similar problems have been proposed recently in the literature. PersonalRAID [19] allows a portable storage devices to be used for propagating updates among several personal replicas. However, this approach makes it impossible for a user to access all his data in a new computer. Footloose [9] introduces the concept of physical eventual consistency, allowing portable devices to be used to automatically propagate updates amongst replicas. FEW extends the approach of Footloose by allowing clients to obtain data contents from other replicas (even outside the system), thus minimizing the requirements of the mobile devices. Also, we allow multimedia data to be transported efficiently.

EnsemBlue [12] supports the integration of data created in consumer devices into a common namespace, transcoding data based on application needs, using what the authors describe as *persistent queries*. FEW uses a similar approach

during the synchronization process for adapting multimedia contents based on the specifications of the user.

Unmanaged Internet Architecture [2] allows users to provide personal names to their devices and data, providing users access this data, from any device, using these names. Perspective [16] and Cimbiosys [15] provide replication solutions, allowing users to keep data replicas in multiple devices, based on the semantic description of that data. Contrarily to our system, these have no mechanism to efficiently store data in mobile devices, other than partial replication.

New cloud storage services (e.g. Dropbox) offer functionalities similar to *traditional* distributed file systems. While these minimize the complexity of setting up file servers, their use is not fully transparent (since data needs to be stored under specific directories), also requiring users to trust on third-party organizations. Additionally, these services are usually only used to store a small subset of users' files. Our system can integrate these services for improving data availability.

6 Final Remarks

FEW is a data management system designed to allow users access their personal data in any machine, independently of location and network connectivity. To this end, FEW has been designed to take advantage the storage capacity and wireless communication capabilities of current mobile phones, for maintaining replicas of the users data (the most up-to-date version). The optimistic replication approach allows for long lived replicas to be created, and accessed, whenever and wherever needed. For guaranteeing consistency, we have proposed novel techniques for tracking dependencies among replicas that can improve the scalability of commonly used version vectors.

FEW addresses the limitations of mobile devices, in particular of the storage capacity of mobile phones. A Data Transcoding module allows FEW to deal with the volume of multimedia data stored by current users, reducing storage requirements by storing lower fidelity versions of these files. A Data Source Verification module allows the system to record alternative sources for the files stored in a container. This approach explores the common case where the files stored were obtained from the Web, or are stored in some remote server. To our knowledge, FEW is the first system to combine mobile and cloud storage to provide high availability while reducing the volume of data that needs to be stored and still offering an single view of a container in all devices the users uses. This mechanism help improving the performance of the system, while reducing power consumption by reducing communications with the mobile phone.

By combining both modules, FEW allows clients to always access full-fidelity contents. This is a new feature when compared with previous solutions that use data transcoding. The obtained evaluation results suggest that the combination of DTC and DSV are important for achieving the goals of the system, since considerable percentages of files, stored by the users, are multimedia files and/or have additional Web sources.

References

1. Dropbox: Dropbox (2011), http://www.dropbox.com/
2. Ford, B., Strauss, J., Lesniewski-Laas, C., Rhea, S., Kaashoek, F., Morris, R.: Persistent personal names for globally connected mobile devices. In: Proc. of the 7th Symp. on Operating Systems Design and Implementation, pp. 233–248 (2006)
3. Google: Google docs (2009), http://docs.google.com/, http://docs.google.com/
4. Guy, R.G., Heidemann, J.S., Mak, W., Popek, G.J., Rothmeier, D.: Implementation of the Ficus Replicated File System. In: USENIX Conf. Proc., pp. 63–71 (1990)
5. Hac, A., Jin, X., Soo, J.H.: Algorithms for file replication in a distributed system. J. Syst. Softw. 14(3), 173–181 (1991)
6. Henk, C., Szeredi, M., Pavlinusic, D., Dawe, R., Delafond, S.: Filesystem in Userspace (FUSE) (December 2008), http://fuse.sourceforge.net/
7. Soares, J.: FEW Phone File System. Master's thesis, Faculdade de Ciências e Tecnologia (April 2009)
8. Soares, J., Preguiça, N.: Proving Ubiquitous Access to the User's Data Combining Mobile and Cloud Storage. Tech. Rep. 04/2011, CITI / DI-FCT-Univ. Nova de Lisboa (May 2011)
9. Paluska, J., Saff, D., Yeh, T., Chen, K.: Footloose: a case for physical eventual consistency and selective conflict resolution. In: Proc. Fifth IEEE Workshop on Mobile Computing Systems and Applications, pp. 170–179 (October 2003)
10. Parker, D.S., Popek, G.J., Rudisin, G., Stoughton, A., Walker, B.J., Walton, E., Chow, J.M., Edwards, D., Kiser, S., Kline, C.: Detection of Mutual Inconsistency in Distributed Systems. IEEE Trans. Softw. Eng. 9(3), 240–247 (1983)
11. Pawlowski, B., Juszczak, C., Staubach, P., Smith, C., Lebel, D., Hitz, D.: NFS version 3 design and implementation. In: Proc. of the Summer USENIX Conf., pp. 137–152 (1994)
12. Peek, D., Flinn, J.: Ensemblue: integrating distributed storage and consumer electronics. In: Proc. of the 7th Symp. on Operating Systems Design and Implementation, pp. 219–232 (2006)
13. Phan, T., Zorpas, G., Bagrodia, R.: Middleware support for reconciling client updates and data transcoding. In: Proc. Int. Conf. on Mobile Systems, Applications, and Services, MobiSys (2004)
14. Preguiça, N., Marques, J.M., Shapiro, M., Letia, M.: A commutative replicated data type for cooperative editing. In: Proc. of the 2009 IEEE Int. Conf. on Distributed Computing Systems, pp. 395–403 (2009)
15. Ramasubramanian, V., Rodeheffer, T.L., Terry, D.B., Walraed-Sullivan, M., Wobber, T., Marshall, C.C., Vahdat, A.: Cimbiosys: a platform for content-based partial replication. In: NSDI 2009: Proc. of the 6th USENIX Symp. on Networked systems design and implementation, pp. 261–276 (2009)
16. Salmon, B., Schlosser, S.W., Cranor, L.F., Ganger, G.R.: Perspective: semantic data management for the home. In: FAST 2009: Proccedings of the 7th Conf. on File and Storage Technologies, pp. 167–182 (2009)
17. Satyanarayanan, M.: Fundamental challenges in mobile computing. In: Proc. of the ACM Symp. on Principles of Distributed Computing, pp. 1–7 (1996)
18. Satyanarayanan, M.: The evolution of coda. ACM Trans. Comput. Syst. 20, 85–124 (2002), http://doi.acm.org/10.1145/507052.507053
19. Sobti, S., Garg, N., Zhang, C., Yu, X., Arvind Krishnamurthy, R., Wang, O.Y.: PersonalRAID: Mobile Storage for Distributed and Disconnected Computers. In: Proc. First Conf. on File and Storage Technologies, pp. 159–174 (2002)