

User Controlled Sharing in a Variable Connected Distributed System*

Tage Stabell–Kulø and Terje Fallmyr
Department of Computer Science
University of Tromsø
Norway
Tage@computer.org, Terje.Fallmyr@acm.org

Abstract

This paper describes the design and implementation of a distributed file repository that supports data sharing in a variable connected environment. Our design is based on the observation that it is difficult to make a clean cut between the various choices for important issues like consistency and concurrency control when the system is designed. Hence, we make it possible for the user to choose whether to adopt an optimistic or a pessimistic approach, rather than embedding it into the system; that is, the user is placed in the decision loop. Unlike many contemporary systems, our repository treats the optimistic and pessimistic approach uniformly and allows users to select them on a per-file basis. Furthermore, even if a file has to be managed pessimistically, the advantage of the optimistic approach—high availability—is retained. The file repository has been implemented, and it is our research vehicle.

1. Introduction

The main aim of this paper is to focus on the importance of, and possibility for, controlled user intervention that changes the division of responsibility between the user and the system (or application). In doing so, we will also report on our endeavor to explore these issues.

The design of distributed systems build on a particular set of assumptions. The set is chosen at design time, and remains fixed. For example, there is a system wide division of responsibilities between the user and the system. As a result, there is one single user model and one user-machine model. Users are expected to behave the same, within the rules set by the design. This may not be a good strategy when building systems that integrate portable and mobile

computers due to the heterogeneity in systems encompassing such machines.

There are many reasons why users would like to control more of their applications' and systems' behavior. The main reason is that users have different affiliations with different machines. Contrasted to a traditional workstation, small, personal machines incur new challenges. Small machines have few resources in general, but are usually trusted by their owners. This diversity must be reflected in the system design.

With several machines at the disposal for one application, a user may decide which computer to use in a given situation; the desktop machine with its resources and convenience, or the trusted pocket machine, or one in between. In general, using a desktop machine that is a part of a larger infrastructure and maintained by the technical support staff, means that important decisions regarding for instance security policies and data consistency semantics has already been taken by others. And they remain fixed. As more personal machines are used, users will feel that they have more responsibility in these matters. This feeling will of course differ among users. In order to gain the level of control each user requires, the option to reconsider important policies in the system may be needed, albeit in a controlled manner.

We have designed and implemented a system compassing integrated heterogeneous devices, where important decisions regarding issues like security, data consistency, concurrency control are flexible. The exact granularity of control depends on the user (competence and confidence), what type of machine the user is using, what type of resources are available (networking, battery power, etc.), the machine's affiliation with the user, for which purposes the user trusts the machine, and for which task the machine is used.

In this paper, the focus is on the importance of, and possibility for, controlled user intervention that on-the-fly alters the division of responsibility between the user and the system or application. The users of such a system may choose to rely completely on the default working of the system or may choose to take more control and thereby decide upon

*Published in the Proceedings of the Seventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-98), June 17–19, 1998, Stanford California, USA, pp 250–255.

the policies or mechanisms that is important to the user.

We will not treat issues like efficient hoarding and conflict resolution, or mobile storage utilization in this paper as we regard them to be orthogonal to the issues addressed here.

The rest of this paper is structured as follows. Section 2 gives an overview over the File Repository and explores the problem of user control. Section 3 contrasts our approach to other efforts. The current state of the project is discussed in Section 4, and we conclude in Section 5.

2. System overview

Users have a wealth of extra-system information. It will range from well defined messages such as “Bob is away for the weekend” to more subtle and uncertain information about colleges working habits. It is hard, if not impossible, to compile this knowledge such that it can be utilized by systems. Humans, on the other hand, are able to consider the implications, and act accordingly. For example, if Bob is away for the weekend, he will most certainly not edit any files. In other words, if Alice shares some files with Bob, she can most probably alter them without taking any steps to protect herself against consistency problems. Furthermore, users can normally use communication channels outside the system proper, channels that can overcome network partitions (telephone is the prime example).

To this end a research vehicle has been designed and realized. It has been named File Repository (FR). It is distributed application that supports users with a wide range of machines, from the smallest PDA to most powerful workstation. It has proven to be valuable in the investigation into system support for mobile machines.

2.1. File Repository

Basically, FR stores files and information about files, and is implemented by servers. Users run clients on their machines, and the clients interact with servers by means of a custom designed protocol (FRTP) [9]. One server is able support many clients. With a suitable client, users can store and retrieve files. The server maintains state concerning the whereabouts of files; who has copies on which machines, which version is current, who was granted locks, when locks expire, and so on. Notice, however, that FR does not provide the functionality of traditional source control systems such as RCS or SCCS. The reason is that the “diff” between two files only makes sense when the files are in clear text (spreadsheets are not, for example).

FR supports replication¹ which enables users to share

¹The term *copy* has to do with distribution in space, while *version* implies changes over time. Replication ensures that all copies of a file has the same version, and a *replica* is a copy which is kept up-to-date.

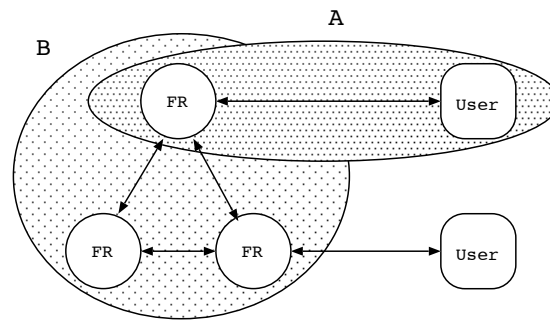


Figure 1. Replication at two levels

files also when they are separated by WANs; replication also increases availability. When a file is replicated, the servers will maintain consistency between the replicas by means of an inter-server protocol [7]; this protocol is based on two-phase commit.

There are two regimes for interaction in the system. The first, labeled “A” in Figure 1, is between clients and their FR. The other, labeled “B” in the figure, is the inter-server communication required for replication (usually over WANs). The latter is outside the reach of users in the sense that when a file is replicated, the FR will exhibit best effort to keep all replicas consistent.

Furthermore, files can also be *shipped* from one FR server (site) to another by a particular variant of the replication machinery. The effect is that a copy of the set of files are sent to the remote server; the sending is done by means of the replication machinery. Shipping files has two applications. First, when the right to access a file is delegated to a user which uses a different (remote) server, the file can be shipped to that server and then subsequently extracted. Second, before departure to a remote site (which runs a FR server), a user can delegate to himself in the rôle of visitor at a remote site, access to his own files locally rather than from the FR at his “home” location. In that way, upon arrival, the files will be present.

A client may lock a file in FR to ensure consistency; locking is the ultimate pessimistic concurrency control regime. Locks are replicated on replicated files. When reading, users might decide to ignore locks based on extra-system information. The semantics is, when a locked file is extracted from FR, a warning is issued. If the user later tries to write data back, the data is relayed onto a shadow of the original file. Thus, a lock can be regarded as a mechanism used to ensure correctness rather than one which prevents users from acting.

There are two more features related to locking. The first is that each lock is identified by what might be called a capability. By giving a capability away, a user can transfer

ownership of a lock to another user. The second feature is that if data is written optimistically on a locked file, the holder of the capability can later release the lock (even without having the file itself).

Denial of service, even though it may preserve correctness, is avoided in our system².

FR runs entirely in user space, and users are free to install one or more FR server on any machine at their disposal. As files can be replicated by users at will, a high degree of availability can be achieved.

2.2. User in the decision loop

During a network partition, it is, in general, impossible to know, from within the system and/or application in the minority partition, whether it is safe to proceed with a task that alters shared data. In fact, it is even impossible to know whether a datum has become shared. The user, on the other hand, may be capable of understanding the situation and *evaluate* the risk, and, more importantly, understand the consequences of acting. Also, as mentioned above, the user might use channels outside the system to gather information.

In a scheme where users have the opportunity to ignore warnings, progress is always possible but can obviously lead to situations where copies of files become inconsistent, situations that it might seem natural to label as incorrect. The FR has machinery to handle such situations. For example, consider the user Alice, who has a computer in her office, on which the latest up-to-date copy of a file is stored; Alice is at home. During the evening she decides to work, but her requests for an up-to-date copy can not be fulfilled due to a network partition, that is, she is unable to contact the FR server in her office. Alice is now left to *decide* which actions the system shall undertake on her behalf. It is obvious that she can safely proceed working at home, if she knows the state of the file, and can refrain from altering those sections where she knows her out-of-date replica differ from the more up-to-date version at work. Exploiting her understanding of the situation she is able to take the “correct” action even though she has created an inconsistency. When the network again becomes operational, the copy in Alice’s office is augmented with a shadow. The shadow contains her updates, and she can either merge the two, replace one of them, or create a new file. The point is, *the system* did not force her to try to circumvent the system services, but rather made it possible for her to obtain progress even when she was in a minority partition. She obtained progress at the risk of conflicting updates, but at her discretion.

The support provided by the system is such that it reflects, to Alice, the fact that she has used her knowledge

²Security related issues aside.

about the situation to act in a certain way. Rather than forcing her to create a new file from which the changes would have been merged into the original, the system is designed to accommodate mobile machines, and thus unexpected situations.

The functionality provided by FR enables one user with several machines to utilize them better. But FR can also be used to enable users to share data. Consider a different scenario: Alice and Bob are sharing a file; they are employed at different sites, separated by a WAN. One fine Sunday afternoon Alice decides to write, contacts her local FR, and request a write-lock on the file. The response is that a copy of the file is available (after all, it is replicated), but the network is partitioned (between the sites of Alice and Bob) no guarantees regarding neither consistency nor correctness can be offered. Alice is left to decide, based on her extra-system knowledge about the habits of Bob and the work of merging possible conflicts. It is obvious that the system can not guarantee her that Bob will not write data to the file, but *she* decides whether to proceed or not. This example highlights the prudence in permitting users to achieve progress at their own risk, when they so desire.

The two examples illustrate an important aspect of FR. There are strong similarities between one single user replicating data on several machines, and several users sharing data. Although FR was designed with mobile machines in mind, it has turned out to be a viable tool also to boost cooperation. The reason is, it seems, that in supporting a set of heterogeneous machines it also can support a diverse user community.

2.3. Implications

Placing the user in the decision loop requires careful system design. Challenges include identifying the critical junctions where sufficient information about the system is available, and then to capture the essence of the situation in such a way that it can be presented. These are (only) engineering problems. In other words, there is no need to (try to) predict neither what users will actually do nor guess what is taking place in other parts of a partitioned network. By promptly leaving such impossible tasks to the user—who will have to ultimately resolve conflicts anyway—the system’s complexity becomes far less daunting. And as a consequence of a less complex system, users can more often grasp the implications of the information the system provides, hence making it feasible to utilize the possibilities.

As has been described, users interact with FR on two occasions in each session. First, when they check a file out of the repository and into their local environment. As part of this operation, information about the user is checked into the FR. Files can be checked out without locking (a simple read of the file), locks can be set, and, depending on replica-

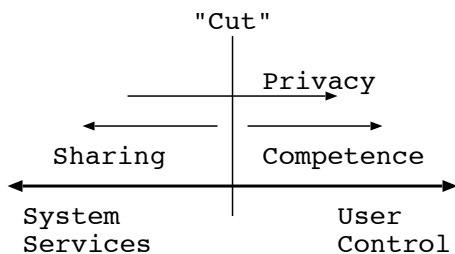


Figure 2. Conflicting interests

tion and connectivity, consistent or non-consistent versions can be read. Second, when changes to a file is to be checked into the repository, files that were not locked might have been changed by others, or, in the case of locked files, shadows might exist.

At an abstract level we can view the inclusion of the “user in the decision loop” as providing the means for the user to decide which actions to take at each junction. The system appears to be dynamic, in that users can choose to have powerful services with fixed “guarantees”, or just be provided with information to facilitate their own decision making.

When faced with choice in a given situation, the result is influenced by at least three conflicting aspects. First, with increased competence and knowledge, the user can discard more and more “supporting” services and move to an operating mode that provides more control. For example, when working alone on a single file, albeit with several machines, the user might be fully aware of when updates have occurred, and where the latest copy is. In this case, very few services are needed. Second, when the complexity increases, for example by more users taking part, so does the need for more stringent concurrency control. Third, users that worry about their privacy may in general shed away from services they do not control in favor of simpler ones. This is illustrated in Figure 2.

The mechanism we have devised for letting the user choose her own level of service ensures that she interacts with FR twice for each operation. These are the junctions where information is traded against service. The first is when she *extracts* files to her current local working space (desktop, notebook, or what have you). Depending on their needs, users can choose, on a per file basis, the semantics regarding concurrency control and consistency, by invoking one of the different options of the operation. Concurrency control options are one of *pessimistic*, which leaves a write lock in FR, or *optimistic* (which just leaves information about the operation).

The second time is when the files are *inserted* again. Logically, when a file is extracted, state is inserted into

the FR, while the subsequent insert operation invalidate the state and updating the file’s content.

There are different options connected to these two basic operations which leaves the user with a set of choices, ranging from waiting for assured consistency in a distributed FR and just dumping a file on the “nearest” server and leave the updating of the replicas to that server. Or, with other words, the user can select a optimistic or pessimistic approach.

3. Related work

The challenges of systems that contain variably connected mobile nodes have been addressed in several projects like Coda [6], Ficus [3], LittleWork [5] and Bayou [10]. Commercial systems and applications are available as well, for example Lotus Notes and Microsoft Access. They address the problems of disconnection and consistency from different points of view, and have found different balances between the conflicts regarding availability, mobile storage utilization, user demand for progress while updating shared data, consistency, and convenience.

System structures vary between client-server and peer-to-peer, and much effort has been put into transparent and efficient hoarding, that is various ways to fill a mobile machine with all the needed data, but no more, for a period of disconnection. Various ways to support re-integration of data after write-write conflicts have been explored too. Other projects have focused on ubiquitous mobile computing and access anywhere, anytime, or focused to provide maximal transparency for the users. Unlike our approach, they all seem to fix the rules for issues like consistency, concurrency, etc., at design time.

Although the objects handled by the repository are *files*, we do not view it as a file system. It does not provide storage, FR manages meta data about the files entrusted to it. Other file systems, notably NFS [8, 11], does not provide storage either. But, in contrast to NFS, FR is not a system service and it is not accessed through system calls. And while NFS must be installed as a file system (in the kernel) in order to catch file system operations as they arrive, FR needs no particular operating system support and it runs entirely in user space. This also makes our system stand out against efforts where system integration is in focus [1].

FR does not enforce optimistic concurrency control as done by Coda. Coda uses conflict detection at reconnect rather than conflict avoidance, as FR makes possible. The fundamental issue is that Coda has no way of knowing whether an inconsistency may occur when the file is modified. Such an approach can not be used in a system where files are frequently shared. Coda assumes that most files are private or part of the system as such—binary programs and libraries that are seldom altered—and manual intervention is needed when conflicts are detected. The FR, on the

other hand, is designed to foster active sharing of files, and write–write conflicts are common.

User control issues have been addressed in other projects, for instance in [2], but we have not been able to find any other projects that address the issues of user control and its consequences for system design, seen in conjunction with both personal mobile machines and personal competence or preferences, which we might call “user heterogeneity”.

4. Current State

FR is first and foremost a research vehicle, providing an environment suitable for experimental research in the field of mobile computing.

For a research vehicle, the performance is satisfactory. To check-in a non-replicated file of 100KB takes 5 seconds. If the file is replicated on five local servers, the check-in takes 13 seconds. With three servers locally, one in The Netherlands (16 hops away) and one in Italy (18 hops away) the same operation takes 23 seconds. All these numbers are almost two years old, and average [7].

Regarding security, we use FR as a non-trivial distributed application where privacy is of great concern. We are trying to get a better understanding of the gap between users and the public key that represents them. We are using palm-top sized machines; an extensive suite of security related software is available. Authentication is currently based on passwords. An infrastructure based on public-key encryption has been designed and is under implementation; under this regime, users are represented by their public keys. The security effort is described in [4]

Both the FR server and client are available for both Unix and Win32, and a library makes it easy to create new clients.

5. Conclusion

We have argued that, to encompass mobile and personal machines in a distributed system, the user of these machines should have the opportunity to be involved in decisions regarding the overall strategies for conflict resolution in the system. We have applied this philosophy to a distributed file repository. Users of the resulting system have control over important aspects of their computing environment, and our file repository is equally well suited to support one user with several machines as several users sharing data. Users are able to take into account their equipment’s different personal affiliation and the heterogeneity in the user community. In particular, users with good understanding of the system are able to work efficiently while novice users get well defined services.

FR provides convenient storage for users with mobile machines, regardless of whether their points of access alternate between home and office, or they travel to remote sites.

The reason our system can provide this range of services, despite its modest size, lies mainly in the possibility for user involvement. However, the user is not continuously engaged, but rather only at two important junctions: conflicts may arise when files are extracted or inserted, the user may choose to get involved in both. The ability to use knowledge effectively and readily, bridges the gap between predicting user behavior—which is hard—and denying service, which is undesirable.

The system we have described consists of two main components, the file repository proper and clients for a variety of equipment. The repository has been implemented and is running on several sites, from Norway in the north to Italy in the south, realizing a distributed repository spanning Europe.

Acknowledgments

This work was done in collaboration with the members of the MobyDick and GDD projects, in Pisa, Twente, and Tromsø. In particular Alberto Bartoli and gianluca dini provided valuable input on an earlier versions on this paper. We thank you all.

References

- [1] M. Bender, A. Davidson, C. Dong, S. Drach, A. Glenning, K. Jacob, J. Jia, J. Kempf, N. Periakaruppan, G. Snow, and B. Wong. UNIX for nomads: Making UNIX support mobile computing. In *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium: August 2–3, 1993, Cambridge, Massachusetts, USA*, pages 53–67. USENIX, Aug. 1993.
- [2] D. Goldberg and M. Tso. How to program networked portable computers. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*. IEEE, October 1993.
- [3] J. S. Heidemann, T. W. Page, R. G. Guy, and G. J. Pop ek. Primarily disconnected operation: Experience with Ficus. In *Proceedings of the Second Workshop on the Management of Replicated Data*. IEEE, November 1992.
- [4] A. Helme and T. Stabell-Kulø. Security functions for a file repository. *Printed in ACM Operating Systems Review*, 31(2):3–8, April 1997.
- [5] L. B. Huston and P. Honeyman. Partially connected operation. In *Proc. of the 2nd USENIX symposium on mobile and location-independent computing*, pages 91–7. USENIX, 1995.
- [6] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

- [7] G. Moxnes. Design og implementasjon av replikering i file repository (in Norwegian). Masters thesis, Department of Computer Science, University of Tromsø, Norway, April 1997.
- [8] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network Filesystem. In *Summer conference proceedings, Portland 1985: June 11–14, 1985, Portland, Oregon USA*, pages 119–130. USENIX, Summer 1985.
- [9] T. Stabell-Kulø. File repository transfer protocol (frtp). Technical report, Department of Computer Science, University of Tromsø, Norway, Feb. 1995. Available as <http://www.cs.uit.no/Lokalt/Rapporter/Reports/9521.html>.
- [10] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing Updates in a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 172–183, Copper Mountain Resort, Colorado, December 3–6 1995.
- [11] D. Walsh, B. Lyon, G. Sager, J. M. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, and P. Weiss. Overview of the Sun network file system. In *Proceedings: USENIX Association Winter Conference, January 23–25, 1985, Dallas, Texas, USA*, pages 117–124. USENIX, Winter 1985.