

# Supporting Mobile Users in a Variable Connected Distributed System: the PASTA Approach\*

Terje Fallmyr      Gunnar Hartvigsen  
Tage Stabell–Kulø

Department of Computer Science  
Institute of Mathematical and Physical Sciences  
University of Tromsø

## Abstract

The PASTA project addresses the consistency problems that occur when data is replicated in a distributed system with mobile machines; characterised by frequent disconnection and varying communication capability. This paper introduces the main problem area for the PASTA project, gives an overview of the solutions offered by the *File Repository* (FR), and presents some initial results. The FR allows users to copy shared data from a distributed file repository onto portable computers before disconnection, and later safely update the data when sufficiently connected. Choice between optimistic and pessimistic concurrency control is offered. Our approach is to provide tools for finding the balance between consistency and desire for progress in a system subject to variable connectivity and write–write conflicts.

## 1 Introduction

With the increasing usefulness of small computers, many users add a portable computer to the set of computers they use to process their data. One person may use a workstation at work, a portable PC while travelling, and a desktop PC at home. This paper addresses our approach to support convenient use of portable (or mobile) computers to read and update shared data.

Even though the portable computers have networking hardware and software, they will not be able to communicate at all times. Moreover, the quality of the offered communication services will vary with several orders of magnitude<sup>1</sup>, and may be expensive to use for longer periods of time too.

Mobile computers therefore need to store local copies of the data they need. To facilitate data sharing, data must be copied onto the mobile computer, and updates must be communicated to be globally visible. Striking the balance between availability and consistency requirements in such systems is critical. This is the crux of the PASTA project.

A straight forward approach may be to copy the needed data (for a day or trip) onto the portable and copy them back upon return. Consistency now relies on manual actions

---

\*This research is done as part of the PASTA project, a joint effort with the Department of Electrical Engineering, University of Pisa, Italy.

<sup>1</sup>e.g., between ATM and modem over GSM [Alanko95]

triggered by the user’s memory of where (on which machine) changes have been done. Files can easily diverge, and manual merging of inconsistent files is painfully slow and error prone.

The system Coda is based on the assumption of a very low frequency of write–write conflicts [Kistler92]. This assumption may be valid in a multi-user setting dominated by read sharing, although the frequency of write-write conflicts in a similar environment are found 16 times higher in [Huizinga94]. Based on experiences in our own environment, we do not share the view that the assumption about very low write–write conflict ratios is valid when one person uses a number of machines to produce changes to the same data. Therefore we seek alternatives to the pure optimistic approach of Coda.

The setting with one user sharing data between several machines, is in essence the same as sharing data between several users. The simple solution described above does not scale to the more general setting with a large amount of files and several machines and users, where the need to automate reads, updates and concurrency control becomes more clear.

The PASTA project seeks simple, yet working solutions, to flexible and scalable support for distributed applications in an environment with variable connected mobile nodes.

We have chosen to base our work on a file abstraction, and for that we use a *File Repository* (FR). Our solution to finding a balance between consistency requirements and desire for progress is based on a user guided selection between pessimistic and optimistic concurrency control. The novel aspect of this work is that, as a result of adopting flexible concurrency control, we obtain a smooth support for disconnected operation and location independent computing. We also obtain a system that nicely supports sharing of data across a wide area network, although that topic is not further discussed in this paper.

The rest of the paper gives an overview of the solutions offered by the FR, and presents some initial results.

## 2 Overview over the File Repository

This chapter gives an overview of the computational model and the architecture of FR.

### 2.1 Computational model

In PASTA, the files to be shared among clients are stored in the *File Repository* (FR). The files are shared according to a check–in, check–out model. FR supports sessions where a user first extracts (checks out) a set of files—that he assumes to be his working set—from the FR and stores it on the client machine he intends to use. The user may then disconnect and work in isolation. Before a new session starts at another machine, modified files are attempted to be reinstalled into the FR.

The model makes a very clear separation between the functions of clients and the FR. The FR always hold the official, globally readable versions of the files. The files copied to clients will still reside in the FR. The FR may be centralised (implemented by one server) or distributed (implemented by more than one server). Files may be replicated in a distributed FR. It is, however, transparent to clients whether the files in FR are replicated, which replication strategy is used, its implementation, and how many servers are involved.

In the machines where clients run, resources are assumed to be scarce. In particular, this is true for communication opportunity and bandwidth. Furthermore, since these machines are under full control of their user, the FR can not depend on them to be neither trustworthy nor available. This has two important effects on the design of our system:

- Since clients can not be trusted by servers, they, and not the server, must take the initiative in any communication. It is the client's responsibility to authenticate itself in order to consume resources at the server.
- Disconnection is the normal mode of operation, and by far the most common situation (most portable and personal are turned off for longer periods than they are turned on). We will therefore not let progress in our system depend on any form of call-back to clients.

In a session, a client can request copies of files and request to write changed files back to the server. Changes made by a client are only visible to the servers and other clients after they have been written back to the FR. Hence, the official version of a file may not always reflect the most recent updates done by clients, and the order of FR updates may not always be the same as the order of the local client updates. This depends on the concurrency control scheme selected by the clients.

## 2.2 Selectable concurrency control

When it is not possible to decide whether it is safe to proceed, there are two alternatives. Optimistic behaviour prescribes to continue work, hoping that no consistency problems will occur. There is progress at the (potential) cost of later having to merge the current changes with that of others. This will for instance be the case when the occurrence a write-write conflict was invisible due to a network partition. The other alternative is to act pessimistically and not change shared data on which there is no exclusive write permission. There will be no inconsistencies, but progress is sacrificed. This is the essence of the well known consistency problem in partitioned networks [Davidson85].

Pessimistic concurrency control is chosen by clients who need a guarantee that write-write conflicts do not occur on files they intend to change. In order to choose pessimistic concurrency control, clients must be able to identify those places where conflicts has a potential to occur. Changed files can be inserted to the FR with the guarantee that no write-write conflict will occur, provided that the insert operation succeeds. If a distributed FR is partitioned when the client reconnects, and the client ends up in a minority partition, the insert operation is not guaranteed to succeed.

The user is responsible for resolving any conflicts that may arise due to insufficient locking. The use of optimistic concurrency control increases availability and promotes progress in distributed computations at the expense of involving the user or application in resolving conflicts. In such cases, user or application level semantics can be used to make decisions that could not be taken at the system level.

## 3 Architecture

The architecture consists of clients that communicate with the FR by the FRTP protocol [StabellKulo95]. Operations available to clients for interaction with the FR are given in Table 1.

Files in FR are immutable to make it easier to handle the problems with concurrent reading and writing. A client that starts a read operation on a file is guaranteed to obtain data from the same version even if a new version of the file is made while the read is in progress. Previous

EXTRACT-P	Pessimistic extraction
EXTRACT-O	Optimistic extraction
EXTRACT-S	Snapshot extraction
INSERT	Ordinary insert
INSERT-A	Asynchronous insert
LOCK	Obtain a write lock
RELEASE	Release a lock
REFRESH	Refresh a lock
FLUSH	Flush file, keep lock
DELETE	Delete file
STAT	File status (meta data)
WALK	Walk directory structure

Table 1: Operations provided by the File Repository

versions are however still available for those clients that hold the appropriate tokens. File versions are not explicitly visible in the naming scheme. A client that presents the name of a file in a read operations, will always get the latest version available.

### 3.1 Pessimistic concurrency control

Clients can use the EXTRACT-P operation to enforce pessimistic concurrency control on a file. A successful EXTRACT-P operation on a file creates a time-limited write lock on the file in the FR, and copies the file and relevant meta data to the client. The meta data serves as a cache for client initiated STAT commands, but also contains a unique, tamper proof *key*, which can unlock the write lock on the file in the FR. After modification, the client may INSERT the file into the FR. Normally, the INSERT operation presents the key to the server in order to certify its write permission, writes the whole file and relevant meta data back to the server and releases the lock. The server will atomically create a new version of the file and associated meta data. Old versions are however guaranteed not to be deleted as long as any operations are still active on them.

### 3.2 Optimistic concurrency control

The EXTRACT-O operation is provided for clients that choose optimistic concurrency control. The operation will return the last committed version of the file. If the file is replicated in FR, the operation will only succeed if a read quorum for the last committed version can be obtained. Otherwise, the client may select the EXTRACT-S operation which copies the last version of the file and meta data available at the server. There is no guarantee about the freshness of that version. The meta data for both the EXTRACT-O and EXTRACT-S operations contain a key that uniquely identifies the file. The LOCK command gives a client that has extracted a file optimistically, either by EXTRACT-O or EXTRACT-S, the opportunity to change into pessimistic concurrency control as achieved by EXTRACT-P.

The operations RELEASE, REFRESH and FLUSH makes it easier to utilise resources based on availability. A write lock may be released with the RELEASE operation without any file data being transferred. The operation will only succeed if the client can show a valid key for

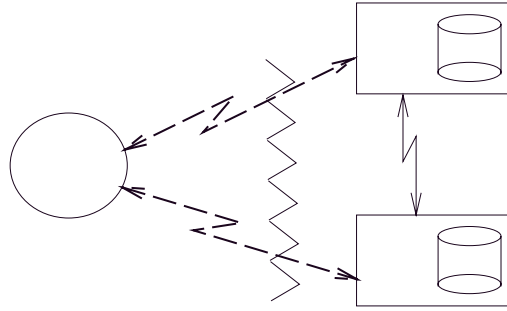


Figure 1: Using different servers

the lock on the file. In the same fashion, the validity of a write lock may be prolonged by the `REFRESH` command.

### 3.3 Servers

The FR is implemented by a set of servers. Servers are normally statically connected computers operating in a stable, well connected environment. In order to ensure progress, the architecture relies on that the servers implementing a replicated FR are not partitioned. However, the architecture does not prohibit that a computer with fewer and varying resources—e.g. a notebook computer—to be a server.

### 3.4 Clients

Client machines are normally portable computers with few resources. Client applications implement user-level operations on the FR and maintains the user’s view of it. They also maintain state, which includes extracted files and corresponding meta data and locks. We make few assumptions about the environments in which these machines are operating.

A client can initiate sessions with different servers, for instance it may check out files from one server and check them in at another, possibly after being disconnected for a long time (see figure 1).

The asynchronous `INSERT-A` operation may be the best choice for a client whose locks are about to expire, and that will accept an operation with less guarantees. The server will propagate the changes (i.e., update replicas) according to a best-effort policy. In this way, the risk of losing changes may be less, or the need for manual merging after the lock has expired may be reduced. Moreover, this allows the client to utilise high bandwidth with a local server without having to wait for updates of replicas, and rely on the server to propagate the changes. These operations require that the user understands their semantics.

## 4 Related work

Although the objects handled by FR are *files*, we do not view the FR as a file system. FR does not provide storage, it manages meta data about the files entrusted to it. Other file systems, notably NFS [Sandberg85], does not provide storage either. But, in contrast to NFS, the FR is not a system service and it is not accessed through the `read` and `write` 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 does not need any particular operating system support and it runs entirely in user space. This also makes the FR stand out against approaches where mobility is handled by changing the operating system, as in [Bender93].

FR provides mechanisms for several concurrency control policies. In particular, FR does not enforce optimistic concurrency control as done by Coda [Kistler91]. The fundamental issue is that Coda has no way of knowing whether an inconsistency may occur, at the time of the update. The user will be faced with this problem later, at reconnect. It is clear that this 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.

We do not strive for transparency, as some systems do [Guy90, Kistler91, Satyanarayanan93]. We believe that providing the means for convenient concurrency control gives the user more flexibility. Although a replica management policy may provide access to data in a seemingly transparent way, the assertion of frequent disconnections will ensure that data will at times be incorrectly assumed up to date. In many settings, we believe it is better to know that update is not safe, rather than relying on some common case optimization which may fail. The important issue is that the user, and not the system, makes the choice when to act optimistically.

Since FR runs in user space, name space maintenance is beyond its scope. Other systems, such as Locus [Walker83] and Coda [Kumar93], provides mechanisms to resolve name space conflicts when the optimistic assumptions is detected to have been violated. FR avoids the problem altogether by the requirement that names are created by a write-all policy at check-in.

Of the three challenges of mobile computing: communication, mobility and portability [Forman94], we do not consider communication technology. Furthermore, we do not consider the effects of physical movement while connected to some network such as dynamic routing, address migration and so forth. We assume that either a machine is connected, or it is not connected.

## 5 Current State

The interface to the FR is realised by the FRTP protocol. This means that any application can utilize the FR “simply” by speaking the protocol. To make life somewhat easier for users and programmers alike, we provide three means to access the FR.

The first is the FR-library that contains a protocol engine. With this library, applications can be written (or modified) to get files from a FR rather than from a local file system. Knowing the name of a file (in the FR) is sufficient to retrieve a copy. The library can both return the contents of the file in a buffer in memory, or given a file handle, write to a local file. In both cases the routines will also return information about the state of the newly obtained copy.

The second is a set of applications. These, which are run in a traditional UNIX command-line style, is not bound to any particular operating system. They contain the FR-library, and uses the widely available Berkeley Socket interface towards TCP/IP on any underlying network. These applications run, in particular, on MS-DOS and UNIX. With these tools,

a user can retrieve a file from the FR and store a copy locally. Any software system, for example a word processor, can then be used on the file as usual. After alteration, the file can be written back to the FR with the same set of tools.

The third is an application, also built on the library, that have the “look and feel” of a file manager. Through it a user can manipulate files in the FR, move files from the local disk or from the repository and onto the local disk, locks can be set and status information obtained. It supports “drag and drop” operations on both files and directories (local and remote), and translates these operations to a set of interactions with the FR by the means of FRTP.

We have implemented a first version of a server that runs on UNIX. Experience suggest re-design of the FRTP protocol to reduce network traffic. Moreover, security is now becoming a concern due to the use of new networking technology. A new server is therefore under development.

Initial results indicate that user selectable concurrency control overcomes the obstacle created by the lack of knowledge of the global state. Users generally have a good idea of what the global state is, and the uncertainty is (partly) overcome by judgement. The FR detects conflicts, but the users’ choice of concurrency control ensures that the system behaves predictable. The preliminary results are based on TCP/IP over Ethernet LAN and modem (SLIP), and, towards remote servers, across the Internet.

We are in the process of establishing a wireless network with small cells based on infrared (IR). IR will play a major role in our testbed for mobile systems in the future. Our partners in Pisa have designed transceivers, and we expect a network to be operative during 1996. We have chosen IR since it exposes us to a number of challenges, most notably a fluctuation in bandwidth and connectivity not found on modern local area networks and the existence of cheap receivers and transmitters available to anyone.

In general, the initial results make us confident that the approach we have taken, combined with the solutions we propose, provide a sound basis for a working experimental system.

PASTA is a basic research project. In order to evaluate some initial design choices, we teamed up with researchers from the University of Twente, The Netherlands, and created a new project, called MOBYDICK. Its focus is to build a system with integrated applications intended for small hand-held computers, a pocket companion. This will enable us to acquire knowledge about the communication and consistency requirements as well as how users actually use the infrastructure we have designed in PASTA. MOBYDICK has been accepted for funding by ESPRIT IV (LTR).

## References

- [Alanko95] Timo Alanko, Markku Kojo, Heimo Laamanen, Mika Liljeberg, Marko Moilanen, and Kimmo Raatikainen. Measured Performance of Data Transmission Over Cellular Telephone Networks. *ACM Computer Communication Review*, **24**(5):24–44, Oct. 1995.
- [Bender93] Michael Bender, Alexander Davidson, Clark Dong, Steven Drach, Anthony Glenning, Karl Jacob, Jack Jia, James Kempf, Nachiappan Periakaruppan, Gale Snow, and Becky Wong. UNIX for nomads: Making UNIX support mobile computing. *Proceedings USENIX Symposium on Mobile & Location-Independent Computing*, pages 53–68. USENIX, Aug 1993.

- [Davidson85] Susan Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, **17**(3):341–70, September 1985.
- [Forman94] G. H. Forman and J. Zahorjan. The challenges of mobile computing. *IEEE Computer*, **27**(4):38–47, April 1994.
- [Guy90] Richard G. Guy, John S. Heidemann, Wai Mak, Jr Thomas W. Page, Gerald J. Popek, and Dieter Rothmeir. Implementation of the Ficus Replicated File System. *USENIX Conference Proceedings* (Anaheim, CA), pages 63–72. USENIX, Summer 1990. Also available from [ftp://shemp.cs.ucla.edu/pub/ficus/usenix\\_summer\\_90.ps.Z](ftp://shemp.cs.ucla.edu/pub/ficus/usenix_summer_90.ps.Z).
- [Huizinga94] Dorota M. Huizinga and Ken A. Heflinger. Experience with Connected and Disconnected Operation of Portable Notebook Computers in Distributed Systems. *Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, USA), pages 119–23. IEEE, Dec. 1994.
- [Kistler91] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda file system. *13th SOSP* (Pasific Grove, Ca, USA 13 October 1991). Published as *SIGOPS*, **25**(5):213–25, October 1991.
- [Kistler92] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *TOCS*, **10**(1):3–25. ACM, February 1992.
- [Kumar93] Puneet Kumar and M. Satyanarayanan. Log-based directory resolution in the Coda file system. *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, pages 202–13, 1993.
- [Sandberg85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network Filesystem. *USENIX Association Summer Conference Proceedings of 1985* (11-14 June 1985, Portland, OR), pages 119–30. USENIX Association, El Cerrito, CA, 1985.
- [Satyanarayanan93] M. Satyanarayanan. Mobile computing. *IEEE Computer*, **26**(9):81–2, September 1993.
- [StabellKulo95] Tage Stabell-Kulø. File Repository Transfer Protocol (FRTP). Technical report CS-TR 95-21. Department of Computer Science, University of Tromsø, Norway, Feb. 1995. Available as <http://www.cs.uit.no/Lokalt/Rapporter/Reports/9521.html>.
- [Walker83] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The LOCUS distributed operating system. *Proceedings of the 9th ACM SOSP*, pages 49–70, oct 1983.