A Model for Multi-Level Consistency

Michael G. Sørensen
{mgsj@diku.dk}
DIKU, Department of Computer Science, University of Copenhagen

February 28, 1998

Abstract

Recently we have started the implementation of a distributed file system that supports applications for mobile computing. It is based on a scheme that uses time as a consistency measure. By allowing applications to specify consistency and modification time bounds they are enabled to *adapt* their behaviour according to the state of their environment (or user demands). They can relax their consistency requirements as the quality of communication decreases (higher cost, higher latency, and/or lower bandwidth) in order to achieve higher availability or reduce cost, and strengthen them again when suited. The scheme allows them to utilize any desired level of optimism or pessimism.

1 Introduction

The advent of mobile computers and the growing popularity of these have inspired many research efforts to integrate and/or support mobile computers within existing distributed systems or even build new ones. The majority of these systems, such as Bayou [10], Coda [8], Ficus [4], and LITTLE WORK [5] have *a priori* decided to use optimistic replica control strategies to achieve high availability for support of mobile computing, even if conflicting updates are unavoidable and the use of stale data is possible. On the other hand (as a bit of an outsider) there is MIo-NFS [2] that uses a pessimistic replica control strategy in order to avoid conflicts, even if this strategy is overly restrictive and cannot be expected to result in high availability.

Since some files (or jobs) are more important than others (to users) and since applications have different file access patterns, why should they all be forced to perform perhaps less than optimal due to the shortcommings of either optimistic or pessimistic replication? What if applications themselves could decide replication strategy and change it dynamically, e.g., due to change in quality or cost of communication or on demand if a user for some reason wishes to override default behaviour? In other words, what if applications could *adapt* their consistency requirements or demands according to the current state of the environment? The need for adaptation, whether on system or application level (or both), has already been recognized as an important [3] or even essential [9] capability of mobile clients (as members of a distributed system).

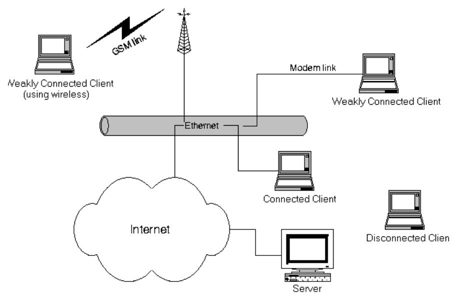
The work described in this position paper is part of the AMIGOS [1] project. The subproject has been nicknamed $\mathcal{P}^e\mathcal{S}^t\mathcal{O}$ for PEssimitic, STrict, and Optimistic, due to the fact that it—in contrast to the systems mentioned above—supports different levels of consistency (and availability). The overall goal is to support mobile computing by enabling applications at all times to *adapt* to their current environment.

1.1 Environment

The system consist of a single (trusted) stationary file server that services multiple (untrusted) mobile clients. For communication we use the `socket` interface. The types of mobile computers supported are discussed in Section 1.2. The service provided is access to shared data (files). The server can be considered as the true home of the shared files, and the mobile clients as caching sites as in the traditional client/server model. The server holds the *primary copy* or the *first-class replica* of the file, and the clients cache *second-class replicas*.

In our environment, a mobile computer can be either connected to (the same network as) the server or disconnected. The connections can be by use of a variety of different technologies, such as Ethernet, dial-up telephone line, dial-up wireless communication, see Figure 1. Hence, the communication bandwidth may vary substantially as does the cost of using this bandwidth. Mobile clients that are connected via a fixed network, with high bandwidth and low latency (e.g., Ethernet) are said to be *(fully) connected*, whereas mobile clients that are connected via other means are *weakly connected*.

Figure 1: System Overview



The mobile clients are in contact with the server on a regular basis, i.e., they do not stay disconnected forever and can be said to be permanent members of the distributed system in which the server resides. The server has no explicit support for stationary workstations, but of course, they can be considered as odd cases of mobile clients, that never move and always are fully connected.

1.2 Mobile Computers

In our environment mobile computers should be *self-contained*, i.e., they should be fully functional computers, with their own operating system (e.g., Windows95, OS/2, or Linux) and applications—allowing the user to work independently from any other machines (e.g., servers). The operating system chosen is Linux, since it is currently in use on the mobile computers used in connection with the AMIGOS project, and it has provision for the use of `sockets`. In the long run it could be any operating system providing a `socket` abstraction (in `C`),[1] so in this sense the mobile computers supported are *heterogeneous*.

One obvious advantage of this choice is that it eliminates the need for caching system files, as in Coda [8], Little Work [5], and SEER [6]. This alleviates the problem of predicting which system files that are actually needed, which is a complicated matter. For example, in Coda [8] a special `spy` program is used to track down use of files during a session and in SEER [6] the problem is solved by constantly logging file references.

2 The Model

Our model is based on the use of time as a consistency measure. With each cached file is associated:

- A Modification Time (MT_{C_i}); the time of the last update to the file,
  - **Note**: From the subscripts it is made clear that the replica resides on the *C*lient (as opposed to on the server), and it is the i'th replica of the file.
- a Consistency Time (CT_i); the time at which the cached file was known to be consistent with the primary copy on the server, and
- a Consistency Check Time (CCT_i); the time of the last check for consistency between the cached file and the primary copy on the server.

If CT_i=CCT_i then the last check for consistency was positive, otherwise negative. A consistency check can be performed simply by comparing the modification time (MT_{C_i}) of the cached file with the modification time (MT_S) of the primary copy on the server.

With every `read` (or any other *non-mutating* operation) must be associated a Consistency Time Bound (*CTB*), and with every `write` (or any other *mutating* operation) a Modification Time Bound (*MTB*). These can either be given explicitly by the application or implicitly using some sort of default value.[2]

[1] It could equally well have been Windows and `winsockets`, but we found it best, at first, to stick with the existing systems.

[2] We have not quite decided how to go about this, yet! Different types of applications or files may require different

2.1 Reading

Let us imagine a client holding the i'th replica of a file, `f`, in the cache with the associated values $<MT_{C_i}$=8.00,CT_i=9.00,CCT_i=9.00$>$. This would be the result if the file was last updated on the server at 8.00, and cached by the client in question at 9.00. At 10.00 (now) the client opens the file for reading:

`open(f,"r",`*CTB*`)`

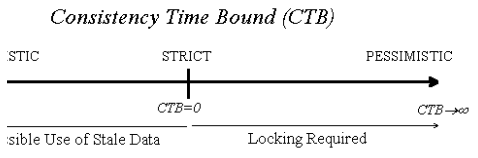
If $CTB>0$ then `f` must be and remain consistent (with the server version) within the specified time bound. For example, with $CTB=2$ hours (from now), then `f` must be guaranteed to be and remain consistent until 12.00. In other words, `f` must not have been updated on the server between 9.00 (where it was cached) and 10.00 (now), and furthermore, it must not be updated for the next 2 hours. If it has been updated between 9.00 and 10.00 then a new copy of the file is required (for it to be consistent now). Under all circumstances a read lock must be obtained (for it to remain consistent within the time bound). If obtaining a new copy of the file or a lock on the file (or both) succeeds then the `open` succeeds, otherwise it fails. Using $CTB>0$ the client is *pessimistic.* The greater *CTB* the more pessimistic.

If, on the other hand, $CTB<0$ then the client is satisfied with a file that was consistent sometime during the period lasting from minimum the time specified by the time bound (ago) and now. In the example above, with $CTB=-2$ hours (from now), the read succeeds because the file was consistent sometime within the last two hours, namely one hour ago (at 9.00). With $CTB=-\frac{1}{2}$ hour, a new consistency check is required because the file cannot be guaranteed to be consistent half an hour ago (at 9.30). If the file has not been updated since 9.00 then the cached file can be used, otherwise a new copy of the file is required (in both cases CT_i and CCT_i can be updated to 10.00). Using $CTB<0$ the client is *optimistic.* The more negative the *CTB* the higher the level of optimism.

$CTB=0$ requires that the file is consistent now, but cannot be guaranteed to remain consistent. By using this *CTB* the client is *strict* in the sense that the file opened should be guaranteed to be consistent with the latest update of the primary copy (at the time of the `open`). The correspondance between the consistency time bound and the level of pessimism or optimism is depicted in Figure 2.

default values.

Figure 2:



2.2 Writing

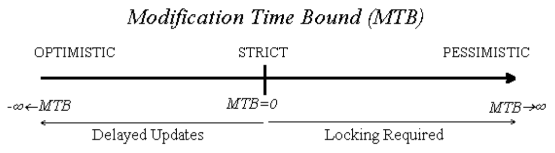
Establishing a connection for every write ensures strict consistency, but cannot be expected to perform well in a mobile computing environment.

The *MTB* associated with an open for writing is the "mutating" counterpart of the *CTB* associated with an open for reading:

`open(f,"w",`*MTB*`).`

A $MTB<0$ means that the writes (done locally) are delayed for a maximum of the specified time bound before written to the server. The delay increases the possibility of write-write conflicts, thus making the write operation increasingly *optimistic* the more negative the *MTB*.

Figure 3:



A write open with a $MTB=0$ assures that the update is propagated to the server as fast as possible, and a $MTB>0$ demands the file to be write locked, see Figure 3. In both cases a connection to the server may have to be established (if it has not already been established or the file is already locked).

2.3 Conflicts

Since optimistic operations are allowed, conflict situations are unavoidable. In the following we imagine two machines (or two different processes on the same machine)—P_1 and P_2—having copies of a file (the newest version) in their caches, and no other machines or processes are accessing the file.

Let us consider a case of concurrent writing; assume for the sake of the argument that the order of operations are: P_1 opens for writing, P_2 opens for writing, P_1 and P_2 close for writing (in an undetermined order). Table 1 shows the possible conflicts.

Table 1: Write/write conflicts

P_2 writing	P_1 writing		
	optimistic	strict	pessimistic
optimistic	FS,SF	FS,SF	P_2 fails
strict	FS,SF	FS,SF	P_2 fails
pessimistic	P_1 fails	P_1 fails	P_2 fails

FS,SF stands for First Succeeds, Second Fails.

Note: When P_1 or P_2 fails, it may not result in write/write conflict, since the `open` rather than the `close` might fail!

What is to be done about write/write conflicts? Some conflicts are more serious than others, and some are easy to fix—but that is totally application-specific, thus conflicts are best handled by applications. How do we inform the applications of conflicts? Write/write conflicts are detected after the file has been closed, either immediately after or after a while, depending on the *MTB*. We choose to let the `close` operation return `SUCCESS` (if no conflicts occur), `FAILURE` (contraversely), or `TIMEOUT`. If it takes some amount of time before the updates are propagated to the server, then it is impossible to report success or failure immediately. How long an application can wait for the close to "finish" is also application-specific, so we will let the applications decide. With each `close` should be associated a close expiration time ($CET \geq 0$):

`close(f,`CET`)`

The `close` will not return until the updates are propagated succesfully back to the server, or a conflict has been detected, or it times out (according to the CET). With CET=0 the call will return immediately, and with a very large CET the call returns only when the status (success or failure) of the `close` has been determined.

If the `close` fails or times out then the application can inform the user, rename the cached file (using a special operation), re-execute the commands, or do whatever steps it deems necessary!

3 Present & Future Work

In order to use communication bandwith optimally and to provide application with the ability to adapt their consistency requirements the implementation will be based on the TACO layer [3]. The file system will use the adaptation facilities provided by TACO, which includes specification of Quality of Service parameters when creating and maintaining a connection, and monitoring of changes in link quality.

At the time of writing the implementation had only just begun, thus we cannot report to you any results. We plan to have a full working implementation with test results no later than at the end of the year!

New (or ported) applications that utilize the new facilities need to be written and their performance weighted against the performance of existing applications before we are able to draw conclusions as to the viability of the model. Furthermore, it will be interesting to study the effects of applications or users operating on the same set of files using different levels of consistency.

Another matter also needs to be resolved; to what level are the applications expected to make their own decisions? Should the system ignore behaviour that seems irrational (e.g., using high degree of optimism, even though fully connected) under the assumption that applications know what they are doing, or should the system try to assist the applications as much as possible?

In the near future we hope to expand the model with a transactional facility, that enable applications or users to group operations into *working units* with the "all-or-nothing" property. Which decisions to make in order to provide an efficient and simple abstraction upon the proposed model for consistency remains (for now) nothing more than an interesting question. This will also make way for detection of read/write conflicts.

There are issues that we have not yet begun discussing but will have to deal with sooner or later. These issues include security (e.g., authentication and encryption), migration (mobility, i.e., crossing network and administration boundaries), and network optimisation (header compression, congestion control and avoidance). By basing our implementation on TACO, some of these issues are solved with the evolution of TACO.

Acknowledgements

The basic ideas for the model presented in this article stem from the report “Transactions in Mobile Computing” [7] by Jeppe Damkjær Nielsen.

Birger Andersen has provided many fruitful comments on all the earlier drafts of this article — Thanks!

References

[1] The **AMIGOS** Project: <http://www.diku.dk/distlab/amigos/>.

[2] Victor P. **Guedes** & Francisco Moura: *Replica Control in MIo-NFS*, ECOOP’95 Workshop on Mobility and Replication, <ftp://ftp.diku.dk/diku/distlab/amigos/mionfs.ps.gz>.

[3] Jørgen Sværke **Hansen** & Torben Reich: *Semi-Connected TCP/IP in a Mobile Computing Environment*, Department of Computer Science, University of Copenhagen, Masters Thesis in Computer Science, DIKU Project no. 95-6-11, 1996.

[4] John S. **Heidemann** *et al.*: *Primarily Disconnected Operation: Experiences with Ficus*, Proc. 2nd IEEE Workshop on Management of Replicated Data, November 1992.

[5] Peter **Honeymann** *et al.*: *The* LITTLE WORK *Project*, Proc. 3rd IEEE Workshop on Workstation Operating Systems, April 1992.

[6] Geoffrey H. **Kuenning**: *The Design of the Seer Predictive Caching System*, IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[7] Jeppe Damkjær **Nielsen**: *Transactions in Mobile Computing*, Department of Computer Science, University of Copenhagen, DIKU Project no. 95-2-11, 1995.

[8] M. **Satyanarayanan** *et al.*: *Experience with Disconnected Operation in a Mobile Computing Environment*, Proc. USENIX Symp. on Mobile & Location-Independent Computing, August 1993.

[9] M. **Satyanarayanan** *et al.*: *Application-Aware Adaptation for Mobile Computing*, School of Computer Sciencce, Carnegie Mellon University, CMU-CS-94-183, July 1994.

[10] Douglas B. **Terry** *et al.*: *Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System*, 15th ACM Symposium on Operating Systems Principles, December 1995.