The Case for Non-transparent Replication: Examples from Bayou

Douglas B. Terry Karin Petersen Mike J. Spreitzer Marvin M. Theimer
Computer Science Laboratory
Xerox Palo Alto Research Center
Palo Alto, CA 94304 USA

Abstract

Applications that rely on replicated data have different requirements for how their data is managed. For example, some applications may require that updates propagate amongst replicas with tight time constraints, whereas other applications may be able to tolerate longer propagation delays. Some applications only require replicas to interoperate with a few centralized replicas for data synchronization purposes, while other applications need communication between arbitrary replicas. Similarly, the type of update conflicts caused by data replication varies amongst applications, and the mechanisms to resolve them differ as well.

The challenge faced by designers of replicated systems is providing the right interface to support cooperation between applications and their data managers. Application programmers do not want to be overburdened by having to deal with issues like propagating updates to replicas and ensuring eventual consistency, but at the same time they want the ability to set up appropriate replication schedules and to control how update conflicts are detected and resolved. The Bayou system was designed to mitigate this tension between overburdening and underempowering applications. This paper looks at two Bayou applications, a calendar manager and a mail reader, and illustrates ways in which they utilize Bayou's features to manage their data in an application-specific manner.

1 Introduction

A major challenge faced by designers of general-purpose replicated storage systems is providing application developers with some control over the replication process without burdening them with aspects of replication that are common to all applications. System models that present applications with "one-copy equivalence" have been proposed because of their simplicity for the application developer [1, 3]. In particular, the goal of "replication transparency" is to allow applications that are developed assuming a centralized file system or database to run unchanged on top of a strongly-consistent replicated storage system. Unfortunately, replicated systems guaranteeing strong consistency require substantial mechanisms for concurrency control and multisite atomic transactions, and hence are not suitable for all applications and all operating environments. To get improved levels of availability, scalability, and performance, especially in widely-distributed systems or those with imperfect network connectivity, many replicated storage systems have relaxed their consistency models. For instance, many systems have adopted an "access-anywhere" model in which applications can read and update any available replica

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

and updates propagate between replicas in a lazy manner [2, 4, 7, 8, 9, 10, 12, 15]. Such models are inherently more difficult for application developers who must cope with varying degrees of consistency between replicas, design schedules and patterns for update propagation, and manage conflicting updates. The harsh reality is that applications must be involved in these difficult issues in order to maximize the benefits that they obtain from replication. The Bayou system developed at Xerox PARC is an example of a replicated storage system that was designed to strike a balance between application control and complexity.

This paper presents both the application-independent and application-tailorable features of Bayou along with the rationale for the division of responsibility between an application and its data managers. Examples drawn from a couple of Bayou applications are used throughout to illustrate how different applications utilize Bayou's features. The applications are a calendar manager and a mail reader. The Bayou Calendar Manager (BCM) stores meetings and other events for individual, group, and meeting-room calendars. A user's calendar may be replicated in any number of places, such as on his office workstation and on a laptop so that he can access it while travelling. Bayou's mail reader, called BXMH, is based on the EXMH mail reader [20]. BXMH receives a user's incoming electronic mail messages, provides facilities for reading messages, and permits the user to permanently store messages in various folders. The BXMH mail database managed by Bayou may be replicated on a number of sites for increased availability or ease of access. Each of these two applications interact with the Bayou system in different ways to manage their replicated data. They demonstrate the need for flexible application programmer interfaces (APIs) to replicated storage systems.

2 Application-independent Features of Bayou

For most replicated storage systems, the basic replication paradigm and associated consistency model are common to all applications supported by the system. While it is conceivable that a replicated storage manager could provide individual applications with a choice between strong and weak data consistency, this made little sense for Bayou. Bayou was designed for an environment with intermittent and variable network connectivity. In such a setting, mechanisms to support strong consistency would not be applicable. Therefore, Bayou's update-anywhere replication model and its reconciliation protocol, which guarantees eventual consistency, are central to the systems architecture. These fundamental design choices over which the application has little or no control are discussed in more detail in the following subsections. Additional application-independent mechanisms for replica creation and retirement are also briefly described. Features that are within an application's control, such as conflict management, are discussed in Section 3.

2.1 Update-anywhere replication model

Bayou manages, on behalf of its client applications, relational databases that can be fully replicated at any number of sites. Each application generally has its own database(s). Application programs, also called "clients", can read from and write to any single replica of a database. Once a replica accepts a write operation, this write is performed locally and propagated to all other replicas via Bayou's pair-wise reconciliation protocol discussed below. This "update-anywhere" replication model, depicted in Figure 1, permits maximum availability since applications can continue to operate even if some replicas are unavailable due to machine failures or network partitions. Thus, it is particularly suitable for applications that operate in mobile computing environments or large internetworks. Because each read and write operation involves a single interaction between a client and a replica, the update-anywhere replication model is also easy to implement and provides good response times for operations.

This replication model was adopted for Bayou because of its flexibility in supporting a diversity of applications, usage patterns, and networking environments [6]. If replicas are intermittently connected, replicas are allowed to arbitrarily diverge until reconciliation is possible. If replicas are few and well-connected, the update-anywhere model is still a satisfactory choice since updates can propagate quickly under such circumstances and

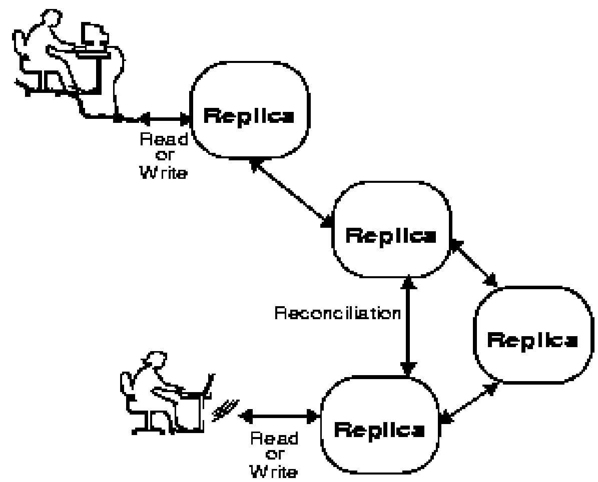

Figure 1: Bayou's update-anywhere replication model.

the replicas remain highly consistent. As described in section 3.1, applications can select reconciliation schedules that best fit their requirements for how much replicas are allowed to diverge.

Consider the example of a user, Alice, managing her personal calendar using BCM. Alice might keep a replica of her calendar on her office machine, one on her laptop, and also one on the office machine of her administrative assistant, Bob, so that her assistant has quick access to her calendar. Alice and Bob's office machines perform reconciliation with each other on a frequent basis so that any updates made to the calendar by either of them are seen by the other with little delay. However, when Alice is travelling, she may update the replica on her laptop while the laptop is disconnected. Any new meetings added by Alice are not readily available to Bob (and vice versa). From her remote destination, Alice occasionally connects to her (or Bob's) office machine via a dial-up modem to exchange recently added meetings, thereby updating their replicas of the shared calendar.

2.2 Reconciliation protocol and eventual consistency

Bayou's reconciliation protocol is the means by which a pair of replicas exchange updates or "writes" [16]. The protocol is incremental so that only writes that are unknown to the receiving replica are transmitted during reconciliation. It requires replicas to maintain an ordered log of the writes that they have accepted from an application client or received from another replica via reconciliation. Pair-wise reconciliation can guarantee that each write eventually propagates to each replica, perhaps by transmission through intermediate replicas, as long as there is an eventual path between a replica that accepts a write and all other replicas. The theory of epidemics indicates that, even if servers choose reconciliation partners randomly, writes will fully propagate with high probability [4]. Arbitrary, non-random, reconciliation schedules can be set up by applications if desired as discussed in section 3.1.

Bayou ensures "eventual consistency" which means that all replicas eventually receive all writes (assuming sufficient network connectivity and reasonable reconciliation schedules) and any two replicas that have received the same set of writes have identical databases. In other words, if applications stopped issuing writes to the database, all replicas would eventually converge to a mutually consistent state. Eventual consistency requires replicas to apply writes to their databases in the same order. Bayou replicas initially order "tentative" writes according to their accept timestamps, and later reorder these writes as necessary based on a global commit order assigned by a primary server [19].

Fortunately, the machinery for managing write-logs, propagating writes, ordering writes, committing writes, rolling back writes, and applying writes to the database are completely handled by the Bayou database managers. Applications simply issue read and write operations and observe the effects of eventual consistency. Applications can optionally request additional session guarantees that affect the observed consistency [18].

2.3 Replica creation and retirement

Bayou permits the number and location of replicas for a database to vary over time. While the replica placement policies are under the control of applications as discussed below in section 3.1, the mechanism for creating new replicas and retiring old ones is application-independent. Bayou allows new replicas to be cloned from any existing replica. The data manager for the new replica contacts an existing replica to get the database schema, creates a local database, and then performs reconciliation with an existing replica to load its database and write-log. Information about the existence of the new replica then propagates to other replicas via the normal reconciliation protocol. This is done by inserting a special ”creation write” for the new replica into the write-log. As this write propagates via reconciliation, others replicas learn of the new replica’s existence [16].

Retirement of replicas is similar. A replica can unilaterally decide to retire by inserting a ”retirement write” in its own write-log. The retiring replica can destroy itself after it performs reconciliation with another replica who will then propagate knowledge of the retirement and of other writes that were accepted by the retired replica.

3 Application-tailorable Features of Bayou

In contrast to the mechanisms for update propagation and eventual consistency, policies and functionalities that vary amongst Bayou applications include how they deal with update conflicts, where they place replicas, and which replicas they access for individual operations. Those issues are discussed in this section. Examples taken from the Bayou applications illustrate how different applications can tailor the Bayou system to meet their specific needs.

3.1 Replica placement and reconciliation schedule

The choice of where to place replicas and when replicas should reconcile with each other is an important policy that is under the control of Bayou applications and users. As described above, the mechanism for replica creation is the same for all Bayou applications. However, the choice of the time at which a replica gets created and the machine on which it resides is completely determined by users or system administrators. The only condition for a replica to be successfully created is that one other replica be available over the network.

Similarly, since Bayou’s weak consistency replication model does not require updates to be immediately propagated to each replica, users are afforded a large amount of flexibility in setting up reconciliation schedules. Experience suggests that such schedules are generally dictated more by the user’s work habits than by the needs of a particular application. For example, a user who works from home in the evening, may wish his office workstation to reconcile with his home machine at 5:00 pm each evening, but does not care about keeping his home machine up-to-date during the day. Also, users and applications often know when are good times or bad times to reconcile with another replica. For instance if the application is in the process of doing a number of updates or refreshing its display, it may not want the database to change underneath it. As another example, a travelling user may dial-in from a hotel room and want reconciliation with the office performed immediately rather than waiting for the next scheduled time.

3.2 Replica selection

Bayou applications generally issue read and write requests without even being aware of which replicas they are accessing. The Bayou client library, which implements the application programming interface (API) and is loaded with the application code, chooses an appropriate replica on which to perform a read or write operation. This choice is based on the availability of replicas, cost of accessing them, and application-chosen session guarantees. The Bayou client library automatically adapts to changing network conditions and replica availability.

Originally, Bayou provided no ability for an application to override the replica selections made by the client library. That is, a Bayou application could not direct its operations to a particular replica. We presumed that most applications, while concerned with the consistency of the data they read, do not wish to be concerned with the specifics of which replicas to access. Moreover, we reasoned that applications do not have enough information about the underlying network connectivity or communication costs to make reasonable decisions about replica selection. What we failed to recognize initially is that users do, in fact, often know quite a bit about the network characteristics as well as the capabilities and consistency of various replicas. For instance, Alice might prefer to access the copy of her calendar that resides on her workstation rather than the one on her laptop, even if the calendar client application is running on the laptop and both the workstation and laptop replicas are available. Hence, users occasionally do want to provide hints about which replicas to access.

Also, there are situations in which an application may want control over replica selection. For instance, an application that supports synchronous collaboration between a number of users, such as a shared drawing tool, may want all these users to access the same replica so that they share the exact same database state. Replication may be desired by this application solely for fault-tolerance, that is, so that it can fail-over to a secondary replica in case the primary fails. Thus, in the second implementation of the Bayou system, we added support for application-controlled replica selection.

3.3 Conflict detection

An inherent feature of Bayou's update-anywhere replication model is that concurrent, conflicting updates may be made by users interacting with different replicas of a shared database. For instance, in the Bayou Calendar Manager (BCM), Alice and Bob could schedule different meetings for Alice at the same time. Such conflicts must be dealt with by each application in an application-specific manner.

The definition of what constitutes a conflict varies from application to application and potentially from user to user. Traditionally, database managers and file systems have pessimistically treated any concurrent writes as conflicting. However, experience with Bayou applications suggest that not all concurrent writes result in application level conflicts. Moreover, writes to separate tuples, which are traditionally viewed as independent, may, in fact, conflict according to the application. Consider BCM which stores each calendar entry or meeting as a separate tuple in the database. Without help from the application, the storage system would detect conflicts as operations that are concurrent at the granularity of either the whole database or individual tuples. If the former, then any concurrently added meetings would be detected as conflicting; if the latter, then no meetings would ever conflict since they involve updating different tuples. Neither of these cases reflect BCM's semantic definition of a conflict.

In BCM, two writes that add new meetings to a calendar or modify existing meetings conflict if their meetings overlap in time and involve the same user(s) or conference room. This simple definition of conflicts is readily supported by Bayou's application-specific conflict detection mechanism. However, we discovered in practice that it did not satisfy all BCM users; some users would prefer to allow overlapping meetings not to conflict and have them scheduled on their personal calendar so they can decide later which meeting to actually attend.

BXMH has a much more complicated model of conflicting operations on a shared mailbox. While BCM basically has a single type of conflict, BXMH has dozens of potential conflict scenarios. BXMH supports 13 operations on a mailbox: adding a new message, moving a message to a named folder, creating a new folder, renaming a folder, deleting a message, and so on. Each of these operations can conflict with other operations in various ways. Moreover, when designing this application, we discovered that potential users could not always agree on which operations conflict under what conditions. The result is that BXMH, through its ”conflict preferences menu”, allows its users to decide what types of concurrent operations should be considered conflicting. Figure 2 shows one of the many conflict scenarios that appears on the BXMH conflict preferences menu. In this example, the user is asked to decide whether moving a message from one folder to another conflicts with a concurrent operation that renamed the destination folder and, if so, how the conflict should be resolved.

Figure 2: Sample conflict scenario from BXMH’s conflict preferences menu.

Although BCM and BXMH have very different notions of conflicting operations, they both rely on the same mechanism to detect their conflicts, namely Bayou’s dependency checks [19]. A dependency check accompanies each write performed by an application. The dependency check is a way for the application issuing the write to detect whether the write conflicts with other concurrent writes. Specifically, a dependency check is a query (or set of queries) and a set of expected results. When the dependency query is run at some replica against its current database and returns something other than the expected results, the replica has detected a conflict; in this case, the replica resolves the conflict, as discussed below, rather than performing the given write. Observe that dependency checks are often specific not only to the application but also to the particular write operation.

For example, if Alice adds a meeting to her calendar from 11 to noon on Friday, BCM creates a dependency check for this write that queries the database for other calendar entries at this time and expects none. Bob might concurrently add a conflicting meeting, say at 11:30 on Friday, because his replica has not yet received Alice’s write. If Bob’s write is ordered before Alice’s, then the dependency check included in Alice’s write will fail.

3.4 Conflict resolution

Strategies for resolving detected conflicts also vary from application to application and user to user. In BCM, a conflict involving two meetings is resolved by trying to reschedule one of the meetings. The meeting that was added last according to Bayou’s write ordering is the one that is rescheduled. In BXMH, the resolution depends on the type of conflict and on the user’s preferences. For example, a user might choose to resolve the conflict in Figure 2 by moving the message to the renamed folder, by leaving the message in its original folder, by creating a new folder for the message or by moving the message to some other existing folder.

Merge procedures in Bayou are the means by which applications resolve conflicts. Specifically, each Bayou write operation actually consists of three components: a nominal update, a dependency check, and a merge procedure [19]. The nominal update indicates changes that should be made to the application database assuming that no conflicting writes have been issued. The dependency check, as discussed above, detects conflicts involving the write. The merge procedure is a piece of application code that travels with the write and is executed to resolve any detected conflicts. The merge procedure associated with a write can query the executing replica’s database and produces a new update to be performed instead of the nominal update. Since merge procedures are arbitrary code, they can embody an unlimited set of application-specific policies for resolving conflicts.

An application is free to pass null dependency checks and merge procedures with each write, in which case the writes issued by the application resemble normal database updates. Importantly, even in the application ignores conflicts, its database is guaranteed to eventually converge to a consistent state at all replicas. Concurrent updates may cause the application not to see some updates because they are overwritten, but eventual consistency is preserved.

3.5 Reading tentative data

Bayou gives applications the choice of reading only committed data or data that may be in a tentative state. The rationale was that some applications may only want to deal with data after it has been committed. Interestingly, the Bayou applications that have been built to date never select the commit-only option when reading data. This is because users always want to see updates that they have made, even if the update has not yet been committed. Bayou indicates which data items an application reads are tentative and which are committed. How the application deals with the information varies with the application. BCM uses this information to show tentatively scheduled meetings in a different color than committed ones. The expectation is that a committed meeting is not likely to change in time, at least not without the meeting organizer informing the participants explicitly, while tentative meetings could get rescheduled due to conflicts. So it is important for the user to know which meetings are tentative and which are not. BXMH, on the other hand, does not distinguish between tentative and committed data when showing a folder's content to the user. The user does not really care whether a particular message is tentatively in a folder as long as the message is successfully displayed when the user clicks on it.

4 Related Work

Early weakly-consistent replicated systems, like Grapevine [2], were intimately tied to particular applications, like electronic mail and name services. The issue of designing replicated storage systems that effectively support a number of diverse applications arose when replication was added to conventional file systems and database management systems. Many of these systems started with the goal of replication transparency but gradually ended up adding hooks for applications to give input to the replication process.

Replicated file systems like Coda [11] and Ficus [17] present applications with a traditional file system interface but also allow them to install "application-specific resolvers" to deal with conflicting file updates. Coda has also recently added "translucent caching" which hides some caching details from users and applications while revealing others [5, 14].

In the database arena, Oracle 7 supports weakly consistent replication between a master and secondary replicas or between multiple masters. It permits applications, when specifying their database schema, to select rules for resolving concurrent updates to columns of a table; each "column group" can have its own conflict resolution method [15]. Applications can provide a custom resolution procedure or choose from a set of standard resolvers.

Lotus Notes, like Bayou, utilizes pair-wise reconciliation between replicas and allows its system administrators to specify arbitrary replication schedules [13]. Notes also permits users and applications to manually invoke reconciliation between two replicas. It detects conflicting updates to a document using timestamps, but has no support for application-specific conflict resolution; alternative versions can be maintained for documents that are concurrently updated.

Bayou, since it was not concerned about backwards compatibility or supporting existing applications, could design a new API that permits more direct application control over various aspects of replication and consistency management. Bayou's conflict resolution mechanism, based on per-write merge procedures, is more flexible than that of other systems, as is its support for application-specific conflict detection.

5 Conclusions

Designing an application programming interface (API) for replicated data is difficult since one must balance the desire for simplicity against the amount of control afforded the application. Simplicity argues for placing common functionality in the replicated storage system, for presenting a storage model that is as close as possible to that of a non-replicated system, and for minimizing aspects of the underlying replication state that are exposed to the application. However, to obtain the maximum benefits from replication, an application needs methods for cooperating with the replicated storage system in the management of the application's data. Permitting such cooperation without requiring the application to assume unnecessary responsibility for the replication process is the key challenge.

The development of Bayou and its applications has allowed us to explore these issues of interaction between applications and replicated data managers. In Bayou, data managers take full responsibility for propagating and ordering updates and ensuring that replicas converge to a consistent state, while applications may control the detection and resolution of update conflicts, create and destroy replicas at convenient times, and set up reconciliation schedules.

Experience building a number of Bayou applications has confirmed the belief that applications need customized control over the replication process. The two applications used as examples in this paper, a calendar manager and a mail reader, have very different policies for detecting and resolving update conflicts. Additionally, they often want different reconciliation schedules. Interestingly, these choices vary not only between applications but also among users of the same application. We conclude that ”replication transparency”, while a laudable goal for supporting legacy applications, is not appropriate for a replicated storage system intended to support a number of applications in diverse networking environments.

6 Acknowledgments

We are grateful for the contributions of our colleagues and interns who have aided in the design and implementation of Bayou and its applications including: Atul Adya, Surendar Chandra, Alan Demers, Keith Edwards, Carl Hauser, Anthony LaMarca, Beth Mynatt, Eric Tilton, Brent Welch, and Xinhua Zhao.

7 References

References

[1] P. A. Bernstein and N. Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. ACM Transactions on Database Systems 9(4):596-615, December 1984.

[2] A. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine: An exercise in distributed computing. Communications of the ACM 25(4):260-274, April 1982.

[3] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in a partitioned network: A survey. ACM Computing Surveys 17(3):341-370, September 1985.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. Proceedings Sixth Symposium on Principles of Distributed Computing, Vancouver, B.C., Canada, August 1987, pages 1-12.

[5] M. R. Ebling. Translucent cache management for mobile computing. Carnegie Mellon University technical report CMU-CS-98-116, March 1998.

[6] W. K. Edwards, E. D. Mynatt, K. Petersen, M. J. Spreitzer, D. B. Terry, and M. M. Theimer. Designing and Implementing Asynchronous Collaborative Applications with Bayou. Proceedings User Interface Systems and Technology, Banff, Canada, October 1997, pages 119-128.

[7] R. A. Golding, A weak-consistency architecture for distributed information services, Computing Systems, 5(4):379-405, Fall 1992.

[8] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. Proceedings 1996 ACM SIGMOD Conference, Montreal, Canada, June 1996, pages 173-182.

[9] R. G. Guy, J.S. Heidemann, W. Mak, T.W. Page, Jr., G.J. Popek, and D. Rothmeier. Implementation of the Ficus replicated file system. Proceedings Summer USENIX Conference, June 1990, pages 63-71.

[10] L. Kalwell Jr., S. Beckhardt, T. Halvorsen, R. Ozzie, and I. Greif. Replicated document management in a group communication system. In Groupware: Software for Computer-Supported Cooperative Work, edited by D. Marca and G. Bock, IEEE Computer Society Press, 1992, pages 226-235.

[11] P. Kumar and M. Satyanarayanan. Supporting application-specific resolution in an optimistically replicated file system. Proceedings IEEE Workshop on Workstation Operating Systems, Napa, California, October 1993, pages 66-70.

[12] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing high availability using lazy replication. ACM Transactions on Computer Systems 10(4):360-391, November 1992.

[13] R. Larson-Hughes and H. J. Skalle. Lotus Notes Application Development. Prentice Hall, 1995.

[14] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. Proceedings Fifteenth ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, December 1995, pages 143-155.

[15] Oracle Corporation. Oracle7 Server Distributed Systems: Replicated Data, Release 7.1. Part No. A21903-2, 1995.

[16] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible Update Propagation for Weakly Consistent Replication. Proceedings 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, October 1997, pages 288-301.

[17] P. Reiher, J. Heidemann, D. Ratner, G. Skinner, and G. Popek. Resolving file conflicts in the Ficus file system. Proceedings Summer USENIX Conference, June 1994, pages 183-195.

[18] D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer and B. B. Welch. Session guarantees for weakly consistent replicated data. Proceedings Third International Conference on Parallel and Distributed Information Systems, Austin, Texas, September 1994, pages 140-149.

[19] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. Proceedings Fifteenth ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado, December 1995, pages 172-183.

[20] B. B. Welch. Customization and flexibility in the exmh mail user interface. Proceedings Tcl/Tk Workshop, Toronto, Canada, 1995, pages 261-268.