Bayou: Replicated Database Services for World-wide Applications

Karin Petersen, Mike Spreitzer, Douglas Terry, and Marvin Theimer

Computer Science Laboratory
Xerox Palo Alto Research Center
Palo Alto, California 94304 U.S.A.
{petersen, spreitze, terry, theimer}@parc.xerox.com

Abstract

The Bayou architecture provides scalability, availability, extensibility, and adaptability features that address database storage needs of world-wide applications. In addition to discussing these features, this paper presents Bayou's mechanisms for permitting the replicas of a database to vary dynamically without global coordination. Key is the use of weak consistency replication among autonomous machines and strict adherence to the tenet that no operation should involve more than two machines.

1. Introduction

Providing scalability, availability, extensibility and adaptability are some of the important challenges faced by designers of world-wide applications and the infrastructure upon which these applications are built. Successful applications running over the emerging global information superhighway must be scalable to support a potentially large, widely-distributed user community. The application's data must be available when any one of the many users needs it. The code base of the applications is likely to evolve and must be able to do so incrementally. The applications must also be adaptable since their demands will be unpredictable and variable. Weakly consistent replication of data among autonomous servers is the key to meeting these demands.

The Bayou system is a weakly consistent replicated database designed to support applications in which distributed users collaborate by reading and writing shared data [11]. Bayou was designed primarily for mobile computing environments where intermittent network connectivity coupled with the desire for high data availability dictates a weak consistency approach to data replication. However, Bayou's design also exhibits good scalability, extensibility, and adaptability since no operation involves more than two machines. In particular:

- Clients can read from and write to any server.
- Servers propagate writes among themselves via a pairwise anti-entropy protocol that permits incremental progress.
- A new database replica, i.e. server, can be created from any existing replica.
- New conflict detection and resolution procedures can be introduced at any time by clients and their applications.

The basic Bayou architecture, including its support for session guarantees and automatic resolution of update conflicts, has been described in previous publications [3][10][11]. The following section focuses on the scalability, availability, extensibility and adaptability features of Bayou. Then we present a more detailed discussion on one aspect of adaptability, namely how to dynamically create and destroy replicas without requiring global consensus among servers. This is a topic that, to our knowledge, has not been adequately discussed in the published literature.

2. Important Bayou 'Bilities

2.1 Scalability

To maximize scalability, the cost of performing an operation at a replica, and the amount of storage required at each replica should be as independent as possible from the number of replicas. Importantly, all Bayou operations, including reading data, writing data, propagating updates between replicas, and creating/destroying replicas involves no more than two computers. Thus the direct cost of these operations is unaffected by the addition of new replicas. By contrast, consider a strongly consistent replicated database that uses a quorum based scheme. Since reads and writes must go to an overlapping quorum of servers, and since writes must be atomically committed at a majority of servers, the operation costs depend critically on the number of replicas. Thus, strong consistency schemes are not attractive for large numbers of replicas or even for small numbers of replicas that are widely distributed and have high inter-replica communication costs.

Bayou's anti-entropy protocol for propagating updates between replicas in a lazy and incremental fashion is also important to its scalability. Importantly, the data exchanged during an anti-entropy session between two servers is simply a pair of version vectors plus any write operations that are unknown to one of the servers. The amount of information transferred does not depend on the overall size of the database. Moreover, this exchange is structured so that the servers move their replicas towards a mutually consistent state even if they become disconnected during the session. Unfortunately, the details of Bayou's anti-entropy protocol cannot be adequately presented within the page constraints of this paper.

With regards to storage costs, each Bayou replica maintains a database and a log of write operations. The database size depends solely on the application's storage needs. Techniques for partial replication, permitting replicas to only store a portion of the database, are being explored but are not included in the current Bayou system implementation. The size of a replica's write log depends on the frequency of updates to the database and the frequency of anti-entropy. This is because replicas are allowed to prune committed writes from their write logs [11].

There is, however, one area in which the Bayou design introduces per-replica storage requirements that depend on the degree of replication, namely in its use of version vectors. Each replica maintains two version vectors that contain an entry for each active replica, one for committed and one for tentative writes. Vectors are exchanged between servers in the first phase of the anti-entropy protocol to determine those writes that have been seen by one server but not the other. Bayou clients also maintain such version vectors to enforce per session consistency guarantees [5]. Fortunately, the size of each entry in these version vectors is relatively small since they simply contain a unique server identifier and a timestamp.

2.2 Availability

Availability in Bayou is maximized by its use of a read-any/write-any replication scheme. That is, any client that can access a server can read and update that server's replica. The primary cost of such availability is the client's need to deal with weakly consistent data and the possibility of conflicting updates. Bayou provides clients with some degree of control over the tradeoffs between consistency and availability through stronger, client-selectable consistency guarantees, called "session guarantees" [10]. These can provide an application with a view of the replicated database that is consistent with its own reads and writes during a session while retaining the principal benefits of a read-any/write-any replication scheme, namely high-availability, simplicity, and scalability. Furthermore, Bayou's support for application-specific conflict detection and resolution enables clients to handle update conflicts in an appropriate manner [11].

2.3 Extensibility

The ability to dynamically extend the functionality of a system is also important when supporting widely deployed applications. Bayou permits such extensibility in the dependency checks and merge procedures used to automatically detect and resolve update conflicts [11]. Since these checks and procedures are passed with each write operation and may vary from write to write, the functionality embedded in them may change as an application evolves. The new functionality is propagated to servers with the writes via Bayou's anti-entropy protocol. Importantly, there is no need to update a group of servers at the same time in order to introduce new conflict resolution procedures.

2.4 Adaptability

Eventually, each Bayou write becomes stable or committed through a primary server commit protocol [11]. Dynamically changing which server is playing the role of the primary is one means of adaptability in Bayou. One may wish to move the primary in response to changing access patterns, for instance, so that the locus of update activity is near the primary server. To become the primary, an existing Bayou server need only contact the existing primary, bring its database up-to-date with respect to the existing primary, and then take on the primary's responsibilities, namely ordering and committing write operations.

As another example of adaptability, the Bayou architecture allows for servers of a database to be created and destroyed over the lifetime of the database. In keeping with Bayou's goal of requiring only pairwise communication between clients and servers and between servers, a Bayou server can be created from any other server and may cease to exist by communicating with only one other server. The next section presents the details of how this is accomplished.

3. Dynamic Creation and Destruction of Servers

3.1 Some Bayou Basics

This subsection provides some background on Bayou, including Bayou's eventual consistency model, ordering of writes, and representation of version vector state. This information is necessary to understand the mechanisms for dynamic replication presented later in the section.

The weak consistency model adopted by Bayou permits database copies at different servers to vary but ensures that each write is eventually received by each server. Bayou is designed so that servers move towards *eventual consistency.* The properties that guarantee eventual consistency are: total propagation of writes by the anti-entropy process, consistent ordering of writes at all servers and deterministic execution of writes [11]. For this section, the ordering of writes is most relevant and therefore discussed in more detail below.

All Bayou servers maintain a log of writes they have received and apply these writes to their databases in the same global order. The position of a write in the global order gets established by its *write-stamp*, a three tuple <*commit-stamp*, *accept-stamp*, *server-id*>. The accept-stamp and server-id are assigned to a write when it is first accepted by a Bayou server from a client. At that point, the write is deemed tentative and its commit-stamp will be set to infinity. Committing a write finalizes the write's position in the global order by assigning the write a unique monotonically increasing commit timestamp. Hence, committed writes are ordered according to the times at which they commit and before any tentative writes.

Bayou also maintains the following write-propagation property, called the "prefix" property: If a server, S_1, holds a write W_x that was initially accepted from a client by another server, S_2, then S_1 will also have received all other writes accepted by S_2 prior to W_x. That is, S_1 will know of all writes accepted by S_2 with an accept-stamp smaller than the one of W_x.

Bayou uses the prefix property to compactly represent the state of a server with a version vector:

WriteVector: a mapping from server identifiers to accept-stamps. WriteVector precisely identifies all the writes known to a server. Specifically, S_1.WriteVector(S_2) is the largest accept-stamp of all the writes known to S_1 that were originally accepted from a client by S_2.

3.2 Managing Dynamic Version Vectors

As Bayou servers learn of new servers or of the retirement of previously known servers, their version vectors grow and shrink accordingly. In a long-lived system, having destroyed servers stop consuming resources in the remaining servers is particularly important. This means that destroyed servers must be eliminated from the version vectors used by Bayou to summarize the writes known to a server. Interestingly the ordering and prefix properties of Bayou writes can be used for this purpose.

A Bayou server S_i creates itself by sending a *creation write* to another server S_k. Any server for the database is sufficient for this purpose. The creation write is handled by S_k just as a write from a client. It receives a write-stamp of the form <infinity, T_{k-i}, S_k>, where T_{k-i} is the accept-stamp assigned to the creation write by S_k, and is inserted in S_k's write log. Once created, S_i can perform anti-entropy with S_k to receive the database contents.

The creation write serves two main purposes. First, as it propagates via anti-entropy, it informs other servers of the existence of S_i. The effect of the write is that an entry for S_i gets added to the server's WriteVector. Second, it provides S_i with a server-id that is globally unique. Specifically, the <T_{k-i}, S_k> component of the write-stamp of S_i's creation write becomes S_i's server-id.

When a server is going to cease being a server for a database, it does so by issuing a *destruction write* to itself. Again, the write is stamped just like any other Bayou write. Its meaning is that the server is going out of service. At this point, the server can no longer accept new writes from clients. However, the server must remain alive until it does anti-entropy with at least one other server so that all its writes, including its destruction write, get propagated to other servers.

When a server receives a destruction write, it removes an entry from its version vector. The prefix property ensures that a server S_k will have received and processed all writes accepted by S_i before removing S_i from its version vector.

One thorny problem remains, however: a creation or destruction write may never reach some servers because servers are allowed to truncate committed writes from their log to save storage resources. For instance, a server may learn of a committed destruction write, process it, and then immediately discard it. Thus during anti-entropy, a server may be presented with a version vector from another server with entries for server-ids that it does not know about, and vice versa.

A server S_i may be absent from another server's version vector for two reasons: either the server never heard about S_i, or it knows that S_i was created and subsequently destroyed. This is the classic create/delete ambiguity. Fortunately, the recursive nature of server identifiers in Bayou allows the server to determine which case holds.

Consider the scenario in which S_1 and S_2 exchange their version vectors during anti-entropy and S_1 has an entry for S_i = <T_{k-i}, S_k> whereas S_2 does not. There are two possible cases:

If S_2.WriteVector(S_k) $\geq T_{k-i}$, then server S_2 has seen S_i's creation write; in this case, the absence of S_i from S_2.WriteVector means that S_2 has also seen S_i's destruction. S_1 can safely assume that server S_i is defunct and remove it from its version vector.

If S_2.WriteVector(S_k) $< T_{k-i}$, then server S_2 has not yet seen S_i's creation write, and thus cannot have seen the destruction either. S_2 should therefore add an entry for S_i to its version vector.

This scenario assumes that S_2.WriteVector includes an entry for S_k. Since multiple servers may go out of service or be created at about the same time, S_2's version vector may be missing entries for both S_i and S_k in the example used above. The presence of an entry for S_k is not essential to identify destroyed servers. Bayou's algorithm is based on the recursive nature of the server identifiers. Imagine a CompleteWriteVector that extends the information stored in the WriteVector to include timestamp entries for all possible servers. A recursive function can compute entries for this extended vector as follows:

CompleteWriteVector(S_i = <$T_{k\text{-}i}$, S_k>) =
WriteVector(S_i) if explicitly available
minus infinity if CompleteWriteVector(S_k) < $T_{k\text{-}i}$
plus infinity if CompleteWriteVector(S_k) $\geq T_{k\text{-}i}$

A value of minus infinity indicates that the server has not yet seen S_i's creation write, and plus infinity indicates that the server has seen both S_i's creation and destruction writes. The recursion terminates correctly as long as the entry for the oldest server in the system never gets removed from the version vector. A server can use its CompleteWriteVector as defined above to correctly resolve differences between the set of replicas represented in its version vector and those known to another server.

In summary, the Bayou write ordering and propagation properties enables the system to handle creation and deletion of servers with the same pairwise communication paradigm as all other Bayou operations.

4. Related Work

A number of systems have been designed based on weak consistency replication among peers that process read requests locally and propagate writes in a lazy fashion. Hence, they achieve the same basic scalability and availability properties of Bayou. These systems include Grapevine [1], Clearinghouse [2], Locus [12], Ficus [4], and Lotus Notes [5]. In particular, all of these systems provide better scalability and availability than replicated systems based on strong consistency. They all differ from Bayou along several axis: use of version vectors, management of conflicts, and consistency seen by clients.

The Locus project at UCLA pioneered the use of version vectors [8], which have been adopted for use in many other replicated systems. However, we are not aware of any systems in which the size of a version vector adjusts dynamically in response to changes in the set of replicas. This adaptability could prove to be very important for world-wide applications.

With regards to conflict detection and resolution, systems like Coda [6] and Ficus [9] require resolvers, the equivalent of Bayou's merge procedures, to be installed in each server. Thus, changing or adding resolvers to evolve or extend an application's functionality is a fairly heavyweight operation. Bayou is the first system we know that permits the rules for application-specific conflict detection and resolution to vary for individual write operations.

While Bayou's session guarantees have not been the focus of this paper, it is worth noting that they provide applications with intermediate levels of consistency without sacrificing the desirable scalability properties [10]. The lazy replication work from MIT comes closest to matching Bayou's replication and consistency features [7].

5. Conclusions

The Bayou replicated database system has managed to achieve good scalability by strict adherence to the tenet that no operation should involve more than two machines. This design principle also leads to other properties that are necessary for world-wide applications, including availability, extensibility, and adaptability. Some operations, such as reading and writing data, involve a single client and any server. Operations such as exchanging writes between replicas and creating new replicas occur between two server machines. Some functions, such as detecting and resolving conflicting updates and applying writes in a consistent order can be done locally by a server without further communication. The key to Bayou's design is the use of weak consistency replication among autonomously operating servers.

6. Acknowledgments

We thank our many colleagues for their valuable contributions to the Bayou system design, especially Alan Demers, Carl Hauser, and Brent Welch.

7. References

[1] 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.

[2] 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.

[3] A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch. The Bayou architecture: Support for data sharing among mobile users. *Proceedings IEEE Workshop on Mobile Computing Systems & Applications*, Santa Cruz, California, December 8-9, 1994, pages 2-7.

[4] 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.

[5] 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.

[6] P. Kumar and M. Satyanarayanan. Flexible and safe resolution of file conflicts. *Proceedings USENIX Technical Conference*, New Orleans, Louisiana, January 1995, pages 95-106.

[7] 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.

[8] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering* SE-9(3):240-246, May 1983.

[9] 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.

[10] 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.

[11] 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*, Cooper Mountain, Colorado, December 1995, pages 172-183.

[12] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel. The LOCUS distributed operating system. *Proceedings Ninth Symposium on Operating Systems Principles*, Bretton Woods, New Hampshire, October 1983, pages 49-70.