PATENT OWNER

EXHIBIT 2034

Designing and Implementing Asynchronous Collaborative Applications with Bayou

W. Keith Edwards, Elizabeth D. Mynatt, Karin Petersen,
Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{kedwards, mynatt, petersen, spreitzer, terry, theimer}@parc.xerox.com

ABSTRACT
Asynchronous collaboration is characterized by the degree of independence collaborators have from one another. In particular, collaborators working asynchronously typically have little need for frequent and fine-grained coordination with one another, and typically do not need to be notified immediately of changes made by others to any shared artifacts they are working with. We present an infrastructure, called Bayou, designed to support the construction of asynchronous collaborative applications. Bayou provides a replicated, weakly-consistent, data storage engine to application writers. The system supports a number of mechanisms for leveraging application semantics; using these mechanisms, applications can implement complex conflict detection and resolution policies, and choose the level of consistency and stability they will see in their databases. We present a number of applications we have built or are building using the Bayou system, and examine how these take advantage of the Bayou architecture.

KEYWORDS: computer-supported cooperative work, asynchronous interaction, distributed systems, Bayou.

INTRODUCTION
Collaboration involves sharing: the sharing of data, artifacts, context, and ultimately ideas. The CSCW community has often categorized collaborative systems based on the temporal aspect of sharing: applications in which users share some "thing" at the same time are called synchronous. Applications in which the users share that thing at different times are called asynchronous.

Synchronous applications, typified by such systems as ShrEdit [15][18] and SASSE [1], are highly-interactive, "real-time" systems in which a group of possibly distributed users interact together to achieve some result. Much of the recent research into collaboration, with the exception of electronic mail [7] and occasionally group editing studies [17] has focused on new tools and techniques to support synchronous collaboration.

 Published in *Proceedings of Tenth ACM Symposium on User Interface Software and Technology* (UIST'97), Banff, Alberta, Canada. October 14-17, 1997.

Asynchronous systems, however, present a number of unique challenges to designers and builders of collaborative systems, from both the human and the technological perspectives. Asynchronous systems are appealing because they allow their users to manipulate time and space to their own advantage—users can work when and where they please, without being constrained by the schedules or locations of others. This style of work, and the settings where asynchronous systems are deployed, have implications for the design of infrastructure and applications. Asynchronous systems must accommodate groups of largely autonomous users, perhaps only loosely connected to each other at any given time.

This paper explores design issues for collaborative systems in general, and asynchronous systems in particular. We examine the reasons that users opt for asynchronous interaction, and the implications of those choices for designers of collaborative infrastructure and applications. We also present a system, called Bayou, designed to support data sharing by groups of individuals working together.

Bayou is an infrastructure for supporting distributed and collaborative applications in which all user interaction involves reading and writing a shared, replicated database. Unlike many infrastructures for collaboration, Bayou is capable of operating over a range of connectivity parameters, from high-bandwidth and constant connectivity, to low-bandwidth and only occasional or unreliable connectivity, as in the case of mobile users. Bayou is a true distributed system—meaning that there is no single centralized location at which data is stored—with weak consistency among replicated data.

Bayou provides mechanisms for application builders to describe the semantic constraints of their applications to the system. These mechanisms allow applications to supply their own data-integrity constraints, conflict detection and resolution procedures, and data propagation policies.

In the following section, we discuss some of the characteristics of asynchronous work, and the properties of asynchronous work that make it desirable for many forms of collaboration. Next, we examine the impact of these characteristics on infrastructure and application design—of necessity, any system for supporting asynchronous work must be informed by the properties of such work.

Then, we describe the Bayou infrastructure. We detail the goals of the system, how it works, and the implications of Bayou for application builders. To demonstrate how Bayou supports the design of asynchronous systems, we describe a set of applications built on top of Bayou. These applications span a range of complexity and interactivity, and each presents a set of lessons for infrastructure builders and application writers.

CHARACTERIZING ASYNCHRONOUS COLLABORATION

Asynchronous collaboration is typically characterized as "different place/different time" collaboration. This characterization is often too simplistic, however. For many asynchronous systems, the defining characteristic is not the fact that the collaboration *doesn't* happen at the same time, rather that it *needn't necessarily* happen at the same time. This distinction is not simply a pedantic one—it has implications for designers of applications and infrastructure.

In an asynchronous setting, the reason that collaboration can happen at different times is because the users do not need to coordinate with one another interactively, and do not need to be notified in "real time" of each other's changes to the artifacts they are sharing. Certain collaborations may lend themselves to this style of interaction because of the nature of the task itself, the work practices of the participants, or the state of the technology at hand.

Tasks that are suitable for this style of work often require little interactive coordination and sharing of work. Collaborators typically can work independently for periods of time, and there is little need for instantaneous propagation of results.

Work practices that favor asynchrony are characterized by people exploiting time and space to work at their convenience and with limited disruption. Such practices may come about because of setting (time zones that prevent collaborators from working at the same time, for instance), or personal desire (minimization of interruption by letting telephone calls "roll over" to voice mail for example).

Technological constraints may also favor asynchrony. Common examples of these include limited network bandwidth that prevents fine-grained or timely sharing of information, and disconnected use (such as using a laptop on an airplane) that separates collaborators.

Independence is perhaps the key trait of asynchronous work. In asynchronous interaction, collaborators, while still operating on some shared set of data, context, information, or artifacts, do so largely independently of one another.

In such work, the need for coordination—communication *about* the collaboration—is lessened, or at least less frequent than it is in synchronous work. For example, collaborative paper writing—at least in the non-computer mediated case—typically involves fairly infrequent coordination. Authors work largely independently, "syncing up" only when necessary to integrate results, or to reaffirm goals or plans [17].

Further, asynchronous tasks that center around some shared artifact do not typically require that all participants immediately know about changes to that artifact. In fact, in some cases such knowledge may be detrimental because it disrupts individual efforts and may incur coordination overhead, when such operations may be more profitably deferred to later.

SUPPORTING ASYNCHRONOUS COLLABORATION

The properties of tasks, work practice, and technology that lend themselves to asynchronous interaction point to infrastructure traits that can support applications for asynchronous tasks.

Independence points to the need to "insulate" collaborators from the actions of others—collaborators should be able to operate with limited interference from or coordination with others. In particular, they should be able to *continue* working, regardless of the actions taken by coworkers. Replication of data is often a useful means for achieving independence of work. Replication can separate the actions of users from their colleagues, providing performance, fault-tolerance, and the ability to locally integrate changes before releasing them to the world at large.

One of the strongest forms of independence is the ability to work completely disconnected from the network and, by implication, other users. The desire to support disconnected use means that users must be able to view, update, and add to their own private replicas of data even when they are not on the network. This constraint requires us to support replicas that are only weakly consistent with one another. If we required strong consistency then all parties would have to be connected at all times, and users would lose a degree of independence from one another.

While eventual consistency of replicas is desirable, users also need to control when information is shared with other users. Applications such as word processing or software development might require explicit control over information propagation. For example, in the case of collaborative software development, users often wish to ensure that updates are withheld until a complete, coherent, and stable picture of the code is available.

Finally, since asynchronous interaction often relies on the fact that collaboration can be achieved even in the face of minimal coordination among users, support for automatic resolution of conflicts can help reduce the need for coordination. If we can mechanically deal with conflicts, we can relieve uses of the burden of "by hand" coordination about their shared artifacts. To be usable by a range of applications, the conflict facilities must be able to implement application-specific policies about how to deal with conflicts. Succinctly, applications must be able to provide their own semantics about how to resolve conflicts automatically.

In the following section we describe a system called Bayou that satisfies these requirements for supporting asynchronous collaboration.

BAYOU OVERVIEW

Bayou is a replicated, weakly consistent storage system designed to support collaborative applications in distributed computing environments with varying network connectivity [22]. A typical example of such an environment is a system with mobile hosts that may disconnect over periods of time, connect only through low-bandwidth radio networks, or connect occasionally with expensive cellular modems. Its model for replication and weak consistency—allowing disconnection of servers from the network—is designed to support extreme scalability, up to "world wide" applications. Bayou relies only on pair-wise communications between computers, which allows the system to cope with arbitrary network connectivity.

Bayou applications can read from and write to any available replica without the need for explicit coordination with other replicas. Every replica eventually receives updates from all other replicas through a chain of pair-wise exchanges of data. To handle the update conflicts that naturally arise in such a weakly consistent system, Bayou allows applications to specify how to detect and resolve these conflicts. In addition, Bayou allows applications to select or specify a number of other policies that control how and where read and write operations get executed.

These characteristics make Bayou well suited for building wide-area asynchronous collaborative systems.

The Bayou System Model

In Bayou, replication is managed by Bayou servers. Each server holds a complete replica of the data. The data model provided by the current implementation of Bayou is a relational database, although other data models could be used as well. We chose a relational model because of its power and flexibility. In particular, it naturally supports fine-grained, structured access to the data, which is useful for the application-specific conflict detection and resolution mechanisms described below. Higher-level application-defined data constructs can be created in terms of the data model provided by the relational database.

As mentioned above, Bayou replicas are weakly consistent. That is, at any point in time different servers may have seen different sets of updates and therefore hold different data in their databases. Weak consistency distinguishes Bayou from many of the replicated systems designed in the CSCW community [3][10]. Some collaborative and distributed systems infrastructures use fairly strong forms of consistency, usually based on pessimistic locking. That is, before data can be modified it must be locked to ensure that its access is serialized. Such strongly-consistent schemes ensure that applications always see a consistent picture of the data. However, they do not support weakly-connected applications, and do not scale to the global applications envisioned by Bayou.

Much like Lotus Notes [13], Bayou applications are free to read and update replicas at will, without locking. Bayou guarantees that the distributed storage system will move toward eventual consistency by imposing a global order on write operations and by providing propagation guarantees. Each write carries enough information so that a Bayou server can apply the writes it has received in the correct order without coordinating with any other server.

Bayou's Mechanisms for Application Semantics

One feature that distinguishes Bayou from previous replicated storage systems including Ficus [12], Coda [14][21], and Lotus Notes [13] is that applications can impose their own semantics on the operations executed at a replica. To this end, Bayou reads and writes are not the simple operations supported by most databases. Instead they include additional application-supplied information, which ensures that applications will receive the required level of service from the system.

Bayou's mechanisms for supporting application semantics fall into six categories:

- Application-defined conflict detection.
- Application-defined conflict resolution.
- Selection of session guarantees.
- Selection of committed or tentative data.
- Replica selection.
- Selectable anti-entropy (data propagation) policies.

Conflict Detection and Resolution. The first two semantic categories are provided through the Bayou write operation, and are designed to detect and resolve the conflicts that arise in a weakly-consistent system. In Bayou, a *write* consists of three components:

- Dependency Check
- Update Set
- Merge Procedure

The *dependency check* specifies a set of conditions that must hold so that the *update set* can be applied to the replica's database. A dependency check consists of a query to be performed at the database and the expected result of that query. If the actual result matches the expected result, then the update set in the write is applied to the database. The update set consists of insertions, deletions, or modifications of tuples in a relation.

If the dependency check fails, an application-specific conflict has been detected and the merge procedure is executed. The *merge procedure*, or "mergeproc" in short, is a fragment of code in a high-level interpreted mergeproc language intended to generate an alternate update set to be applied to the database. Mergeprocs support application-defined conflict resolution, meaning that conflicts are essentially handled through application code, even though that code is executed by the Bayou infrastructure itself. We shall see some examples of mergeprocs in our discussion of applications.

Bayou's use of mergeprocs differs from systems like Coda [14][21] and Ficus [12], which also support application-

supplied conflict resolution, in that Bayou allows different resolution procedures to be associated with each individual write. Thus, Bayou provides applications with more fine-grained control over conflict handling. Furthermore, because the conflict resolution procedure propagates with the write it is available at each server when needed.

The mechanisms for automated conflict detection and resolution are important for supporting asynchronous collaboration, because they eliminate situations where users would otherwise be required to interact closely when faced with data conflicts. Hence, Bayou allows users to act more independently.

Session Guarantees. The session guarantees mechanism is used by an application to establish a required level of consistency for its own operations. That is, while a set of Bayou servers maintain data that is only weakly-consistent, a running instance of an application can request that its view of the world maintain a particular level of consistency. Different applications may have different requirements for their desired level of consistency, and Bayou supports a range of applications needs through this mechanism.

A *session* is an abstraction for a sequence of reads and writes performed during the execution of the application, and session guarantees are implemented by constraining the replicas that may be selected by the application during that session.

Four session guarantees are supported by Bayou:

- *Read Your Writes* ensures that the effects of any writes made within a session are visible to later reads within that session. In other words, reads are restricted to replicas of the database that include all previous writes in the session.
- *Monotonic Reads* permits users to observe a database that stays up-to-date over time. It ensures that reads are only made to database replicas containing all writes whose effects were seen by previous reads within the session.
- *Writes Follow Reads* ensures that traditional write/read dependencies are preserved in the ordering of writes at all servers. That is, at every replica of the database, writes made during the session are ordered after any writes whose effects were seen by previous reads in the session.
- *Monotonic Writes* says that writes must follow previous writes within the session. In other words, a write is only incorporated into a replica's database copy if the copy includes all previous writes from that session, and the write is ordered after these previous writes.

Session guarantees are described in more detail in [23], and are not intended to ensure atomicity or serializability. Instead, users of collaborative applications use session guarantees to maintain a self-consistent view of the database, even though they may read from and write to various, potentially inconsistent, replicas over time.

Stable vs. Tentative Data. Bayou provides a mechanism that establishes when a write is *stable* at a given server. That is, when no new writes will ever be received by the server that will have to be ordered before that write. When a write becomes stable at a server, its conflict detection and resolution mechanisms will not be executed again, which means that its final effect on the database is known. On the other hand, a write that is not yet stable at a server is deemed *tentative*. Tentative writes may need to be re-executed if other writes with earlier write-stamps are received by the server, and thus have a possibly changing effect on the database.

The distinction between tentative and stable data is important from the application's perspective. An application can be designed with a notion of "confirmation" or "commitment" that corresponds to Bayou's notion of stability. For example, color codes can be used in a graphical user interface to indicate whether a displayed item is tentative, that is, may change later because of conflict, or is stable and will not change due to conflict.

Bayou also allows clients to choose whether they will read from the database when tentative data has been applied, or only from the view of the database that corresponds to applying only stable writes. This ability allows clients to trade data availability for assurance of data stability—applications that can tolerate data that has not fully stabilized can read it immediately, without waiting for it to become stable.

Although stability does not equate with consistency, when a collaborative application reads only the results of stable writes, its users will perceive a different "sense" of consistency than if the application also reads tentative data.

Replica Selection. Another important feature that Bayou provides to an application is the ability to select which replica it will use for its operations. The ability to select from several replicas over the life-span of an application is particularly important to collaboration:

- A particular replica can be selected to optimize certain communication requirements. In particular, autonomous users with a disconnected laptop can run a server for a local replica on that laptop. Applications can choose this server, thus ensuring access to the database.
- Applications operating on behalf of different users on different machines can be connected to the *same* replica, which enables all the application instances connected to that replica to see updates as soon as they occur. In essence, the applications can work together in a tightly-integrated, strongly-consistent, synchronized fashion. The ability of applications to connect to a single replica, and later split apart and communicate with different replicas, can be used to support transitions between synchronous and asynchronous styles of collaboration.

Anti-entropy Policies. Anti-entropy is the pair-wise process by which the servers of two replicas bring each other's databases up to date. During the anti-entropy process two servers exchange the sets of writes known to one server but

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.