Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System

Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer and Carl H. Hauser

SOSP 1995

Motivation

- Users working concurrently can introduce conflicts for collaborative applications
- Unreliable network connections and high per-minute connection costs mean it is impractical to have continuous connections
- Goal: balance the realistic expectations of poor network connectivity with the desire to maintain weakly consistent, replicated data

Solution: Bayou

- Application-specific determination of how to detect and resolve conflicts
  - Accomplished via dependency checks and merge procedures
- Maintain two states of an update: tentative and committed
- Manage the states to determine when an update can be changed from tentative to committed
- Designed for non-real-time collaborative applications (e.g. meeting room scheduler, mail databases, etc.)
- Ensure that replicas will be eventually consistent

Application Examples

- Meeting room scheduler
  - For a given room, any time for a meeting for that room can be assigned to no more than one person
  - User is presented current state of scheduler with respect to his/her copy of the room schedule (which may be out of date if, e.g., it is a local version that has not recently been updated)
  - As the connectivity allows, the schedule is periodically re-read, and any are updated on user's graphical user interface
- Bibliographic database
  - Users add papers to the database as they are found
  - Must assign unique key to each paper
  - Keys are tentative until committed, since there could be a conflict, and uniqueness of keys must be ensured
  - Cannot expect to maintain persistent connection with library network due to, e.g. student hackers

Layout of Bayou's System Model

- Data collection fully replicated among servers
- Applications communicate with servers through Bayou API
  - Basic operations: read and write
- Access to one server is enough for client to perform read/update operations
- Bayou write operations contain additional information for how to deal with conflicts
- Each write has a globally unique WriteID
- Storage system maintains ordered log of writes
- Writes are passed from one server to another via pairwise contacts ("anti-entropy")
  - Theory promises that, if some assumptions about no servers being permanently partitioned are met, then write will eventually make it to every server

Dealing with Conflicts

- How conflicts are managed is application-specific
- Dependency checks
  - Used to determine if write-write conflict occurs
  - For each write, a query is associated with it which is checked against current state of connected server
  - If database evaluates SQL-like query as holding true, this indicates a conflict
    - Leads to merge procedure
- Merge procedure
  - Also left as a choice of developer
  - Main objective: when faced with a conflict, how to resolve it
  - Room scheduler example: provide a list of multiple preferred times

```
Bayou_Write(
    update = {insert, Meetings, 12/18/95, 1:30pm, 60min, "Budget Meeting"},
    dependency_check = {
        query = "SELECT key FROM Meetings WHERE day = 12/18/95
            AND start < 2:30pm AND end > 1:30pm",
        expected_result = EMPTY},
    mergeproc = {
        alternates = {{12/18/95, 3:00pm}, {12/19/95, 9:30am}};
        newupdate = {};
        FOREACH a IN alternates {
            # check if there would be a conflict
            IF (NOT EMPTY (
                SELECT key FROM Meetings WHERE day = a.date
                AND start < a.time + 60min AND end > a.time))
                    CONTINUE;
            # no conflict, can schedule meeting at that time
            newupdate = {insert, Meetings, a.date, a.time, 60min, "Budget Meeting"};
            BREAK;
        }
        IF (newupdate = {})   # no alternate is acceptable
            newupdate = {insert, ErrorLog, 12/18/95, 1:30pm, 60min, "Budget Meeting"};
        RETURN newupdate;}
)
```

Figure 3. A Bayou Write Operation

Maintaining Consistency

- Eventually consistent
  - Guarantees that all servers eventually receive all writes
- When write is first accepted, marked as tentative
  - Write is associated with timestamp
- Server maintains log of writes, ordered by timestamp
  - Committed writes are ordered before tentative writes
  - Global order of tentative timestamps ensures agreement for isolated cluster of nodes
  - In some cases, tentative writes need to be undone and re-executed
- Need to determine write is stable so that it can be committed
  - Accomplished using a primary commit scheme
    - One server, designated as primary, has the authority to commit updates
    - Desirable because requiring majority quorum is difficult in face of connectivity issues
    - If primary fails, client can still perform useful read and write operations
      - In this case, writes just remain tentative

Implementing the Storage System

- Design of Bayou leads to certain requirements for the structure of the storage system
- Write log
  - Contains write obtained by a given server
  - Writes are in their global committed or tentative order (based on timestamp)
- Tuple store
  - Implemented as relational database
  - Provides support for SQL-like queries
  - Interesting property: maintains two views of data
    - Committed view: only shows committed writes
    - Full view: shows both tentative and committed writes
- Undo log
  - Allows server to undo effects on tuple store

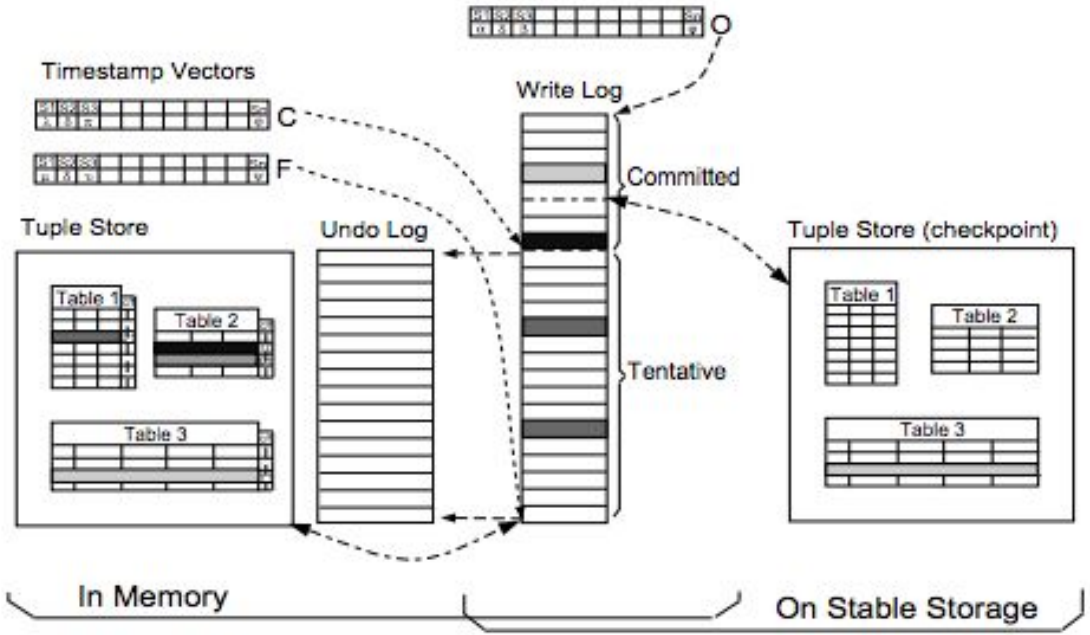

Figure 4. Bayou Database Organization

Access Control

- Because of poor network connectivity assumption, cannot rely on trusted central authentication server
- Instead, mutual authentication is implemented
  - Based on public-key cryptography
  - Uses a single trusted signing authority to sign all certificates
- Authorization is granted for entire data collection
  - Can be revoked if it is found that certificate has been revoked

Performance

- Size of storage increases as number of tentative writes increases
  - Primarily attributed to increase in overhead cost of access control certificate for tentative writes
- Execution time for Bayou server to undo/redo all tentative writes
  - Cost of redoing is nearly constant time
- Performance of operations between client and server
  - In case of conflict, write is not unique
    - Thus, requires additional time for reassignment within merge procedure

Table 1: Size of Bayou Storage System for the Bibliographic Database with 1550 Entries
(sizes in Kilobytes)

Number of Tentative Writes	0 (none)	50	100	500	1550 (all)
Write Log	9	129	259	1302	4028
Tuple Store Ckpt	396	384	371	269	1
Total	**405**	**513**	**630**	**1571**	**4029**
Factor to 368K bibtex source	1.1	1.39	1.71	4.27	10.95

Table 2: Performance of the Bayou Storage System for Operations on Tentative Writes in the Write Log
(times in milliseconds with standard deviations in parentheses)

Tentative Writes	0	50		100		500		1550	
	Server running on a Sun SPARC/20 with Sunos								
Undo all	0	31	(6)	70	(20)	330	(155)	866	(195)
(avg. per Write)		.62		.7		.66		.56	
Redo all	0	237	(85)	611	(302)	2796	(830)	7838	(1094)
(avg. per Write)		4.74		6.11		5.59		5.05	
	Server running on a Gateway Liberty Laptop with Linux								
Undo all	0	47	(3)	104	(7)	482	(15)	1288	(62)
(avg. per Write)		.94		1.04		.96		.83	
Redo all	0	302	(91)	705	(134)	3504	(264)	9920	(294)
(avg. per Write)		6.04		7.05		7.01		6.4	

Table 3: Performance of the Bayou Client Operations
(times in milliseconds with standard deviations in parentheses)

Server Client	Sun SPARC/20 same as server		Gateway Liberty same as server		Sun SPARC/20 Gateway Liberty	
Read: 1 tuple	27	(19)	38	(5)	23	(4)
100 tuples	206	(20)	358	(28)	244	(10)
Write: no conflict	159	(32)	212	(29)	177	(22)
with conflict	207	(37)	372	(17)	223	(40)

Strengths

- Eventual consistency guarantees, via pairwise “anti-entropy” communication
- Flexibility for how application developers manage conflict
- Managing conflict occurs at granularity of per-iteration
- Design is such that servers do not need nearly perfect synchronized clocks
    - Good for unreliable networks

Weaknesses

- Application-specific conflict detection and resolution means more work for the application developer, and is possibly prone to human error
- Potential for cascading conflicts (a new write depends on a previous conflict write, etc.)
    - Mitigated by circumstances of application, e.g. if application is not continuously subjected to (possibly conflict-inducing) writes

Conclusion

- Bayou provides a weakly consistent storage system for mobile applications connecting via an unreliable network
- Conflicts are resolved (or, if unresolved, written to an error log to be dealt with by hand) at the granularity of each individual write
- Bayou maintains both a full and a committed view of the data
- Best suited for situations with low likelihood of conflicts occurring

Questions?