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

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

Presentation by Eric Newberry
Motivation

- Mobile environments not conducive to constant connectivity
  - Especially in mid-1990s...
- Can we create a DFS that can read/write when replicas segmented?
- How do we resolve conflicts once replicas reconnect and resync?
- How do we reorder conflicting writes once a global order is established?
Enter Bayou!

Figure credit: D.B. Terry et al. 1995
The Gist of Bayou

- Clients send read/write request to reachable replica
  - “Write” request can insert, append, or delete
- Replicas store operations in ordered log
  - Writes are “tentative” until global order established (“committed”)
- When replicas reconnect, resolve conflicts between tentative writes
  - This is the complicated part - stay tuned for more!
Detecting and Resolving Conflicts

- Application semantics must be taken into account
  - Only applications can define best way to resolve conflicts
- Solution: Include custom scripts in each write operation
  - Clients specify how conflicts detected and resolved
- Detecting conflicts: “Dependency checks”
- Resolving conflicts: “Merge procedures”
- No locking because of conflicts
  - Contemporary DFS’s required manual unlocking after conflicts
Dependency Checks

- Application query plus expected output
- If output differs -> conflict!
- If conflict, perform “merge procedure”
Dependency Checks

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;}
)
Merge Procedures

- Fix writes so they no longer conflict
- May perform additional read operations
  - Meeting room scheduling app: check if alternative rooms+times work
- Resolve conflict or report error if cannot be resolved
Merge Procedures

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 credit: D.B. Terry et al. 1995
Ensuring Consistency between Replicas

- Bayou’s promise: All replicas will eventually be consistent:
  - All replicas record same committed writes
  - All replicas record committed writes in same order
  - All conflicts resolved deterministically
- Sync writes via periodic “anti-entropy” sessions
- Replicas may achieve tentative consistency between themselves
- Tentative writes must be reversible
  - Consensus re-established as replicas communicate
- How to determine when write is “stable” enough to commit?
Write Stability

- Could use complex method requiring timestamp syncing
- Consider write stable when timestamp before last sync time with all replicas
- But, clocks can drift between hosts
  - See Cristian’s algorithm
- Would need to sync time with ALL replicas
Write Stability

- Instead, designated replica ("primary") commits writes
- All writes tentative until committed by primary -> define global order
- Try to commit in order of write timestamps, but not requirement
- Knowledge of commits propagated via anti-entropy
Replica Databases

Figure credit: D.B. Terry et al. 1995
Access Control

● Coarse-grained access control
  ○ Read, write, and/or replica privileges on *entire* data collection

● Public-key cryptography for replica/client authentication

● Access control certificates for each permission on data collection
  ○ Read, write, act as replica
  ○ Signed by a centralized CA trusted by all
  ○ Permissions can be delegated to others
Authentication and Access Control Exchange

- C: Establish connection to other replica/client
- S: Authenticate client using challenge/response protocol
- C: Sends access control certificate(s)
- S: Check access control certificate(s) against revocation list
- No need to reauthenticate during session
  - If certificate revoked mid-session, session will terminate
- Primary will verify access control certificate for write before committing
Prototype Implementation

- Written in C
- Deployed on Linux and SunOS (yes, this is an old paper!)
- Uses ILU RPC package from then-Xerox PARC
- Multicast-based protocol used for registration and replica lookup
- Dependency checks and merge procedures written in Tcl
Example Applications

- Distributed meeting room scheduler
  - Double-bookings resolved via merge procedures (check alternate rooms/times)
- Bibliography database
  - Dependency checks look for books with duplicate key or duplicate authors+title
  - Merge conditions merge duplicate records
Evaluation: Tentative Writes vs. Data Structure Size

![Graph showing the relationship between number of tentative writes and data structure size. The graph includes lines for Write Log, Tuple Store Checkpoint, and Total size. The x-axis represents the number of tentative writes, ranging from 0 to 1500, and the y-axis represents size in KB, ranging from 0 to 5000.]
Evaluation: Time to Undo/Redo All Tentative Writes

![Graph showing time to undo/redo all tentative writes](image)
Evaluation: Client Performance

![Bar chart showing performance metrics for different scenarios.

- SPARC Server+Client
- Laptop Server+Client
- SPARC Server/Laptop Client

The chart illustrates time (in milliseconds) for various operations:
- Read (1 tuple)
- Read (100 tuples)
- Write (no conflicts)
- Write (conflicts)
Conclusion

- Allows writes during extended network segmentations
- Rely on single primary to establish global order (“commit”)
- Periodically sync replicas via “anti-entropy” sessions
- Dynamic scripts detect and handle conflicts
- Overhead is low, especially if replicas converge frequently
How applicable is Bayou these days?

- Bayou is old! (1995 -> older than me!)
- Designed in a different era
  - Mobile connectivity slow, expensive, insecure, and limited in deployment
  - No WEP (standardized in 1997 [1])
  - Not even IEEE 802.11 (aka WiFi)! (also standardized in 1997 [1])
  - First 3G deployment not until 2001 [2]
- Distributed file systems can still become disconnected
  - Clusters of replicas can converge and later reconverge with entirety of DFS
  - Automatically resolve conflicts in application-specific manners
Distributed File Systems Prior to Bayou

- Handle conflicts transparently, ignoring application semantics
- Applications provide custom conflict detection+handling, but only one global handler for each app
- Store only most recent version of record, preventing rollbacks or reordering
- Clients can’t choose whether reads include all or only committed writes
- Use central authentication and access control server
References

[1] IEEE 802.11-1997 standard