Outline

- 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. In ACM Symposium on Operating Systems Principles (SOSP '95)
  - Epidemic algorithms
  - Serverless file system from MSR

http://www.parc.xerox.com/csl/projects/bayou/

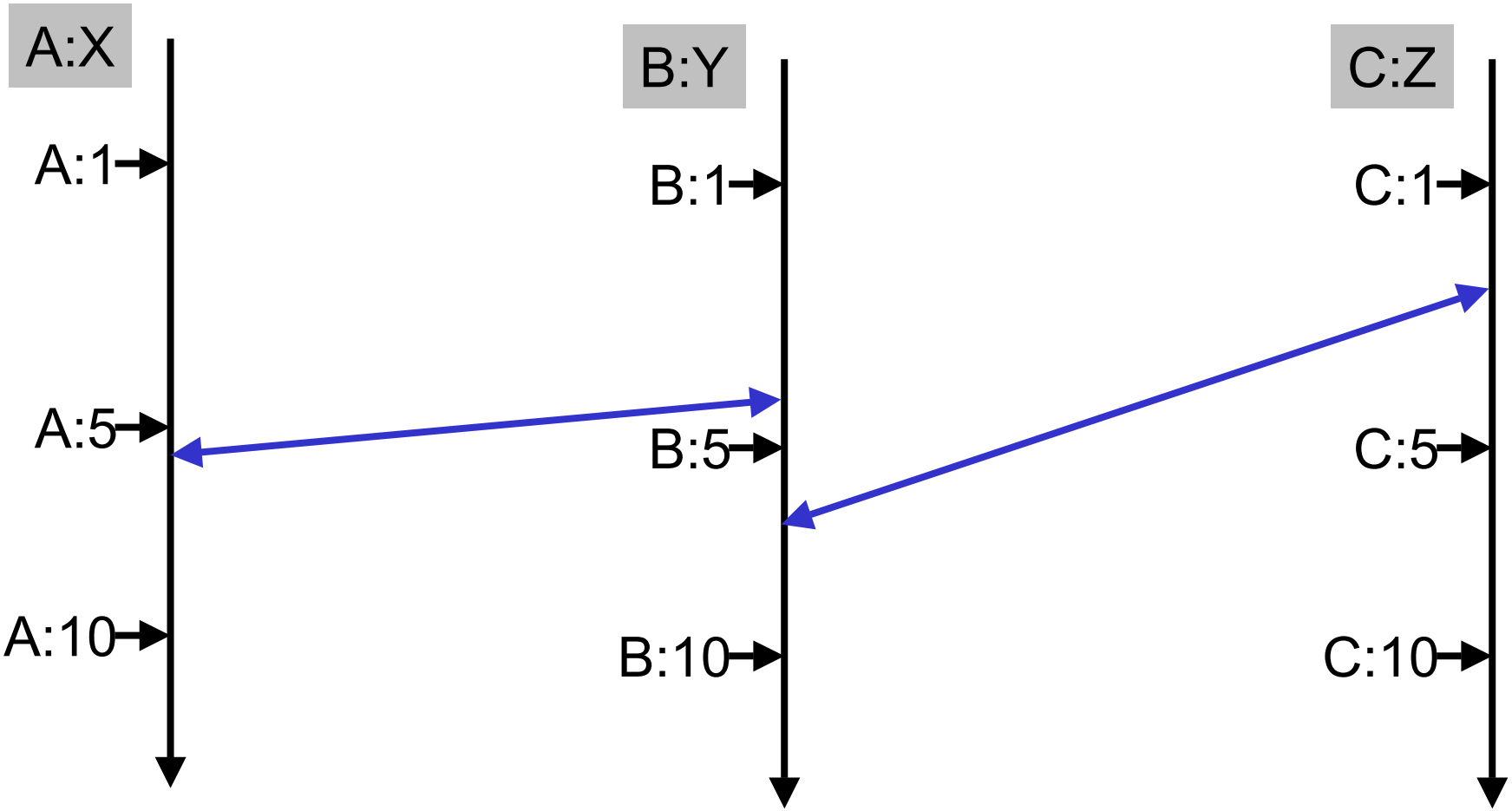
Bayou Write Operation

```
Bayou_Write(update, dependencyCheck, mergeproc)
{
   IF (DB_Eval(dependency_check.query) <>
         dependency_check.expected_result)
    resolved_update = Interpret(mergeproc);
   ELSE
    resolved_update = update;
   DB_Apply(resolved_update);
}
```

Example Scenario

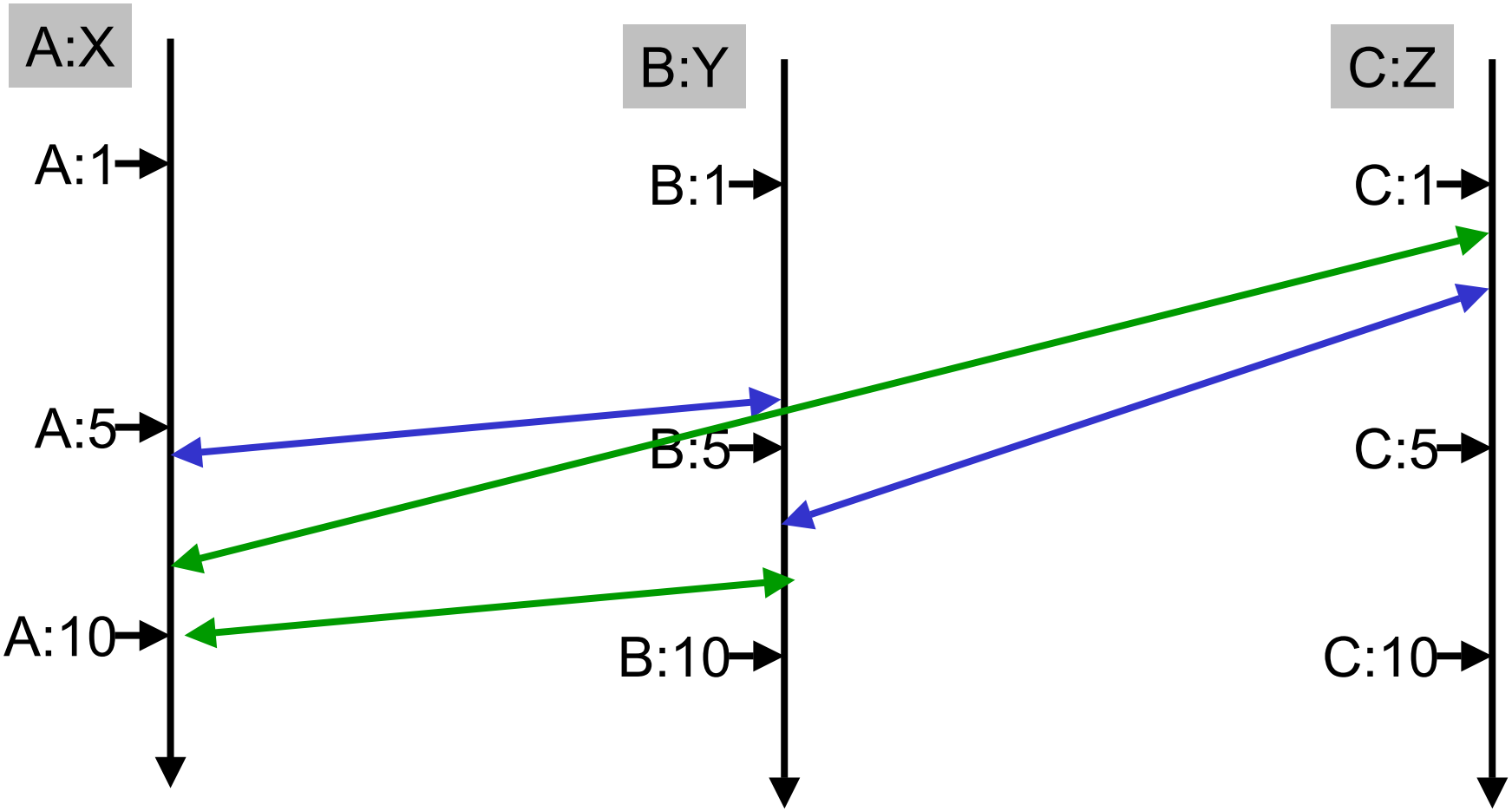
- Three users A, B and C & three replicas X, Y and Z
- They write data items as follows:
  - A:<X,1>, A:<X,5>, A:<X,10>
  - B:<Y,1>, B:<Y,5>, B:<Y,10>
  - C:<Z,1>, C:<Z,5>, C:<Z,10>
- A performs anti-entropy with B at <X,6> and B at <Y,4>. C performs anti-entropy with B at <Y,7> and C at <Z:3>

Example scenario



- B will undo update from A, apply C and then A

Example scenario - Undos



- Support A&C at A:<X,8>, C:<Z,2> & A:<X:10>, B:<Y:8>

Conflict detection and resolution

- Dependency checks allow for application-specific conflict detection
  - Used to detect write-write conflict – two users update the same items without observing the other's update
  - Dependency check for data items that write depended on
- Merge procedures allow for application-specific conflict resolution
  - Merge procedures are specific to each write
  - Bayou allows replicas to be accessible even when mergeprocs fail

Replica Consistency

- Eventual consistency – anti-entropy process
- Write are performed in the same, well-defined order at all servers
- Conflict detection and resolution are deterministic
- It is impossible to know if merge procedures are commutative

Write Stability and Commitment

- Primary commit scheme – One server takes responsibility for committing updates

- Writes from different servers may commit in a different order based on when the servers perform anti-entropy with the primary and each other

Applying Sets of Writes to Database

- Receive_Writes(writeset, received_from) {

```
IF (received_from = CLIENT) {
      logicalclock = MAX(systemclock, logicalclock+1);
      write = First(writeset)
      write.WID = {logicalclock, myServerId};
      write.state = TENTATIVE;
      WriteLogAppend(write)
      BayouWrite(write.update,write.dependency,write.merge)
} ELSE {
    find insertion point;
    rollback till insertion point
    rollback from insertion point
    logicalclock = MAX(logicalclock, write.WID.timestamp);
}
```

Access Control

- Certificates – Grant, delegate and revoke
- Certificates are shared via anti-entropy

Replica Selection

- Users specify expected session guarantee

- Users can choose based on network connectivity, usage patterns etc.

Session guarantees

- *Read Your Writes* - read operations reflect previous writes.
  - Users and applications find it particularly confusing if they update a database and then immediately read from the database only to discover that the update appears to be missing. This guarantee ensures that the effects of any Writes made within a session are visible to Reads within that session. In other words, Reads are restricted to copies of the database that include all previous Writes in this session

Session guarantees

- *Monotonic Reads* - successive reads reflect a non-decreasing set of writes.
  - The Monotonic Reads guarantee permits users to observe a database that is increasingly up-to-date over time. It ensures that Read operations are made only to database copies containing all Writes whose effects were seen by previous Reads within the session.
  - A user's appointment calendar is stored on-line in a replicated database where it can be updated by both the user and automatic meeting schedulers. The user's calendar program periodically refreshes its display by reading all of today's calendar appointments from the database. If it accesses servers with inconsistent copies of the database, recently added (or deleted) meetings may appear to come and go. The MR-guarantee can effectively prevent this since it disallows access to copies of the database that are less current than the previously read copy

Session guarantees

- *Writes Follow Reads* - writes are propagated after reads on which they depend.
  - The Writes Follow Reads guarantee ensures that traditional Write/Read dependencies are preserved in the ordering of Writes
  - Consider a weakly consistent replicated bulletin board database that requires users to post articles or to reply to articles by performing database Writes. The WFRP-guarantee can be used within this system to ensure that users see the replies to a posted article only after they have seen the original. A user who replies to an article must simply issue the reply in the same session as used to read the article being replied to. Users who are only reading articles need not request any guarantees

Session guarantees

- *Monotonic Writes* - writes are propagated after writes that logically precede them.
  - a Write is only incorporated into a server's database copy if the copy includes all previous session Writes; the Write is ordered after the previous Writes.
  - Consider a replicated database containing software source code. Suppose that a programmer updates a library to add functionality in an upward compatible way. This new library can be propagated to other servers in a lazy fashion since it will not cause any existing client software to break. However, suppose that the programmer also updates an application to make use of the new library functionality. In this case, if the new application code gets written to servers that have not yet received the new library, then the code will not compile successfully. To avoid this potential problem, the programmer can create a new session that provides the MW-guarantee and issue the Writes containing new versions of both the library and application code within this session.

Performance

- Size is acceptable
- Write performance is acceptable

Future

- Partial Databases
  - Carry part of the database instead of the entire database (mobile clients do not have enough storage space)

    The problem is that, if a client did not have a particular record, was it because it didn't replicate that part of because it didn't know about it?

    Need some sort of "death certificates"

Discussion