TÉCNICO LISBOA

inesc id lisboa

DAD
Desenvolvimento de Aplicações Distribuídas

Replication – High Availability
Bayou

Bayou

Bayou (1)

- **The Bayou system provides data replication for high availability with weaker consistency guarantees than sequential consistency:**
  - eventual consistency
- **Bayou RMs cope with variable connectivity by exchanging updates in pairs:**
  - The **anti-entropy protocol**
- **Bayou copes with user expectations**
  - **data is always available**
  - **eventual consistency could lead to unbounded divergence**
    - **defines user *session guarantees* for divergence bounding**

Bayou (2)

- **Bayou adopts an original approach:**
  - it enables **domain-specific** conflict detection and conflit resolution to take place.
  - all the updates are applied and recorded at whatever replica manager they reach.
  - When updates received at any two RMs are merged during an anti-entropy exchange, however, the RMs detect and resolve conflicts.
  - Any domain-specific criterion of conflict between operations may be applied.

Bayou (3)

- **Example of conflict resolution in Bayou:**
  - if an executive and her secretary had added appointments in the same time slot, then a Bayou system detects this after the executive has reconnected his laptop.
  - Moreover, it resolves the conflict according to a domain-specific policy.
  - In this case, it could, for example, confirm the executive's appointment and remove the secretary's booking in the slot.
  - Such an effect, in which one or more of a set of conflicting operations is undone or altered in order to resolve them, is called an operational transformation.

Bayou (4)

- The **state** that Bayou **replicates** is held in the form of a **database**, supporting **queries** and **updates** (that may insert, modify or delete items in the database).

- **A Bayou update is a special case of a transaction:**
  - It consists of a **single operation**, an invocation of a 'stored procedure' , which affects several obiects within each replica manager, but which is carried out with the **ACID guarantees**.
  - Bayou may **undo** and **redo** updates to the database as execution proceeds.

Bayou (5)

- **The Bayou guarantee is that:**
  - **eventually**, every RM receives the same set of updates and it **eventually** applies those updates in such a way that the replica managers' databases are identical.
  - In practice, there may be a continuous stream of updates and the databases may never become identical;
    - but they would become identical if the updates ceased.

- **Committed and tentative updates:**
  - Updates are marked as tentative when they are first applied to a database.
  - Bayou arranges that tentative updates are eventually placed in a canonical order (e.g., logical clocks, version vectors, vector clocks) and marked as committed.

Bayou (6)

- **While updates are tentative:**
  - the system **may undo** and **reapply** them as it produces a consistent state.

- **Once committed:**
  - they **remain applied** in their allotted **order**.
  - In practice, the **committed order** can be achieved by designating some RM as the **primary**.
  - In the usual way, **this decides the committed order** as that in which it **receives the tentative updates** and it **propagates that ordering** information to other replica managers. (***C**ommit **S**equence **N**umber*)
  - For the primary, users can choose, for example, a fast machine that is usually available;
    - equally, it could be the RM on the user's laptop, if that user's updates take priority.

Bayou (7)

- **Anti-entropy protocol manages inconsistency and handles update conflicts**
  - Updates **undone** and **redone** as necessary
    - Canonical order may be determined by a primary
  - **Application-specific** conflict detection & resolution
  - **Dependency checks** and **merge procedures**

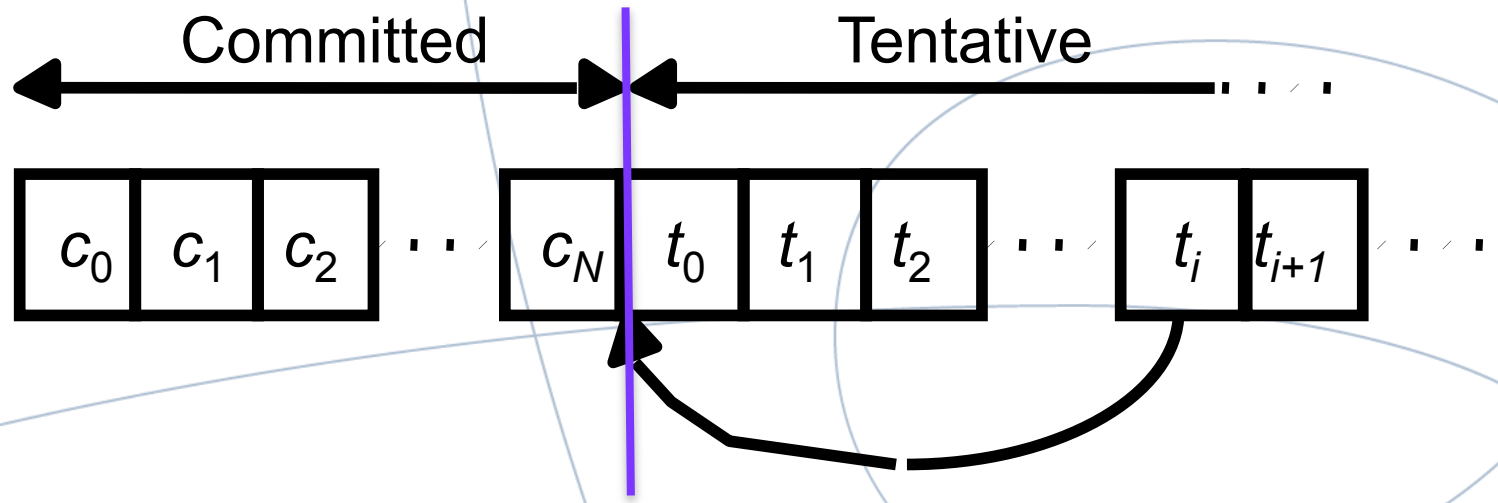
Bayou: dependency checks

- **An update may conflict with some other operation that has already been applied.**
- **thus, every Bayou update contains:**
  - a **dependency check**, and
  - a **merge procedure**
  - in addition to the operation's specification (the operation type and parameters).
  - All these components of an update are **domain-specific**.
- **A RM calls the dependency check procedure before applying the operation.**
  - It checks **whether a conflict would occur if the update was applied** and it may examine any part of the database to do that.
  - E.g. booking an appointment in a diary:
    - The dependency check could, most simply, test for a write-write conflict:
      - that is, whether another client has filled the required slot.
    - But the dependency check could also test for a read-write conflict.
      - it could test that the desired slot is empty and that the number of appointments on that day is fewer than six.

Bayou: merge procedures

- **If the dependency check indicates a conflict, then Bayou invokes the operation's merge procedure.**
  - The merge procedure **alters the operation** that will be applied so that **it achieves something similar** to the intended effect **but avoids a conflict**.
  - E.g., for the diary:
    - the merge procedure could choose another slot at a nearby time instead, or
    - it could use a simple priority scheme to decide which appointment is more important and impose that one
  - The **merge procedure may fail** to find a suitable alteration of the operation, in which case the **system indicates an error**.

Bayou: Committed and Tentative Updates



- **At any time, the state of a database replica derives from a (possibly empty) sequence of committed updates followed by a (possibly empty) sequence of tentative updates.**
- **If the next committed update arrives, or if one of the tentative updates that has been applied becomes the next committed update, then a reordering of the updates must take place.**
- **Figure:**
  - ti has become committed
  - All tentative updates after c_N need to be undone;
  - ti is then applied after c_N and t_0 to ti -1 and ti + 1 etc. reapplied after ti'

Session Guarantees – RYW & MR

- **Session guarantees:**
  - a mechanism to **generate dependencies automatically** from a **user-chosen combination** of the following **predefined policies** (RYW, MR, WFR, MW)
- **"Read your Writes" (RYW) guarantees that:**
  - the contents **Read** from a replica **incorporate previous Writes** by the **same user**
- **"Monotonic Reads" (MR) guarantees that:**
  - **successive reads** by the **same user** return **increasingly up-to-date** contents
  - it ensures that Read operations are made only to replicas containing all Writes whose effects were seen by previous Reads within the session.

RYW Example (1)

- **Example 1 – Read Your Writes**
  - After changing his password, a Grapevine user would occasionally type the new password and receive an “invalid password” response.
  - This annoying problem would arise because the login process contacted a server to which the new password had not yet propagated.
  - It can be solved cleanly by having a session per user in which the RYW guarantee is provided.
  - Such a session should be created for each new user and must exist for the lifetime of the user’s account.
  - By performing updates to the user’s password as well as checks of this password within the session, users can use a new password without regard for the extent of its propagation.
  - The **RYW-guarantee ensures** that the **login** process **will always read the most recent password**.
  - Notice that this application requires a **session to persist across logouts and machine reboots**.

RYW Example (2)

- **Example 2 – Read Your Writes**
  - Consider a user whose electronic mail is managed in a weakly consistent replicated database.
  - As the user reads and deletes messages, those messages are removed from the displayed “new mail” folder.
  - If the user stops reading mail and returns sometime later, she should not see deleted messages reappear simply because the mail reader refreshed its display from a different copy of the database.
  - The **RYW-guarantee** can be requested within a session used by the mail reader to **ensure that the effects of any actions taken**, such as **deleting a message or moving a message** to another folder, **remain visible**.

Session Guarantees – RYW & MR

- **Session guarantees:**
  - a mechanism to **generate dependencies automatically** from a **user-chosen combination** of the following **predefined policies** (RYW, MR, WFR, MW)
- **"Read your Writes" (RYW) guarantees that:**
  - the contents **Read** from a replica **incorporate previous Writes** by the **same user**
- **"Monotonic Reads" (MR) guarantees that:**
  - **successive reads** by the **same user** return **increasingly up-to-date** contents
  - it ensures that Read operations are made only to replicas containing all Writes whose effects were seen by previous Reads within the session.

MR Example (1)

- **Example 1 – Monotonic Reads**
  - A user's appointment calendar is stored 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**.
  - RYW is insufficient as updates are performed by more than a client

MR Example (2)

- **Example 2 – Monotonic Reads**
  - Once again, consider a replicated electronic mail database.
  - The mail reader issues a query to retrieve all new mail messages and displays summaries of these to the user.
  - When the user **issues a request to display** one of these messages, the mail reader **issues another Read to retrieve the message's contents**.
  - The **MR-guarantee** can be used by the mail reader to **ensure that the second Read is issued to a server that holds a copy of the message**.
  - Otherwise, the user, upon trying to display the message, might incorrectly be informed that the message does not exist.

Session Guarantees - WFR

- **"Writes follow Reads" (WFR) guarantees that:**
  - a **Write** operation is **accepted only after Writes observed by previous Reads** by the **same user** are incorporated in the same replica.
  - it ensures that traditional Write/Read dependencies are preserved in the ordering of Writes at all servers.
  - that is, in every copy of the database, **Writes made during the session are ordered after any Writes whose effects were seen by previous Reads in the session**.
  - this guarantee is **different** in nature **from the previous two guarantees** in that it **affects users outside the session**
    - not only does the session guarantees that the **Writes it performs occur after any Writes it had previously seen**,
    - but also all **other clients will see the same ordering of these Writes** regardless of whether they request session guarantees

Session Guarantees – WFR (2)

- **The WFR-guarantee, as defined, associates two constraints on Write operations.**
  - A constraint on **Write order** ensures that a Write properly follows previous relevant Writes in the global ordering that all database replicas will eventually reflect.
  - A **constraint on propagation** ensures that all servers (and hence all clients) only see a Write after they have seen all the previous Writes on which it depends

WFR Example

- **Example 1 – Writes Follow Reads**
  - Imagine a shared bibliographic database to which users contribute entries describing published papers.
  - Suppose that a user **reads some entry**, discovers that it is inaccurate, and **then issues a Write to update the entry**.
  - For instance, the person might discover that the page numbers for a paper are wrong and then correct them with a Write such as “UPDATE bibdb SET pages = ‘45-53’ WHERE bibid = ‘Jones93’.”
  - The **WFR-guarantee** can **ensure** that the **new Write updates the previous bibliographic entry at all servers**.

Session Guarantees - MW

- **"Monotonic Writes" (MW) guarantees that:**
  - a **write** operation is **accepted only after all write operations** made by the **same user** are incorporated in the **same replica**
  - i.e. a **Write is only incorporated** into a server's database copy **if the copy includes all previous session Writes**
  - this guarantee provides assurances that are **relevant** both to the **user of a session** as well as to **users outside the session**

MW Example (1)

- **Example 1 – Monotonic Writes**
  - The **MW-guarantee** could be used by a text editor when editing replicated files to **ensure** that if the user saves version N of the file and later saves version N+1 then version N+1 will replace version N at all servers.
  - In particular, it **avoids** the situation in which version N is written to some server and version N+1 to a different server and the versions get propagated such that version N is applied after N+1.

MW Example (2)

- **Example 2 – Monotonic 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.

TÉCNICO LISBOA

inesc id lisboa

Ensuring Session Guarantees (1)

- **These guarantees are sufficient to solve a number of real-world problems.**
- **Session guarantees are implemented using a session object carried by each user (e.g., in a personal mobile device)**
- **A session records two pieces of information:**
  - the **write-set** of past write operations submitted by the user, and
  - the **read-set** that the user has observed through past reads.
  - each of them can be represented in a compact form **using vector clocks**.

Ensuring Session Guarantees (2)

- **The implementations require only minor cooperation from the servers that process Read and Write operations.**
- **Specifically, a server must be willing to return information about:**
  - the **unique identifier (WID)** assigned to a new Write,
    - **e.g., <procID, clock>: locally monotical clock may be physical, logical counter,**
  - the **set of WIDs** for Writes that are relevant to a given Read, and
  - the **set of WIDs** for all Writes in its database.
- The burden of providing the guarantees lies primarily with the **session manager** through which all of a session's Read and Write operations are serialized.
- The **session manager** can be considered a **component of the front-end** (client stub) that mediates communication with available servers.

Ensuring RYW

We define DB(S,t) to be the ordered sequence of Writes that have been received by server S at or before time t.

- **Providing the Read Your Writes guarantee involves two basic steps.**
  - Whenever a **Write** is accepted by a server, its assigned WID is added to the session's write-set.
  - Before each **Read** to server S at time t, the session manager must check that the write-set is a subset of DB(S,t).
- **This check**
  - could be done **on the server** by passing the write-set to the server, or
  - could be done **on the client** by retrieving the server's list of WIDs.
- **The session manager can continue trying available servers until it discovers one for which the check succeeds.**
  - If it cannot find a suitable server, then it reports that the guarantee cannot be provided.

Ensuring MR

- **Providing the Monotonic Reads guarantee is similar to RYW:**
  - **before** each **Read** to server S at time t, the session manager must ensure that the read-set is a subset of DB(S,t).
  - **after** each **Read** R to server S, the WIDs for each Write in RelevantWrites(S,t,R) should be added to the session's read-set.
  - This presumes that the server can compute the relevant Writes and return this information along with the Read result.

Ensuring WFR

- **Writes Follow Reads guarantee can be provided as follows:**
  - As with Monotonic Reads, each **Read** R to server S at time t results in RelevantWrites(S,t,R) being added to the session's read-set.
  - Before each **Write** to server S at time t, the session manager checks that this read-set is a subset of DB(S,t).

Ensuring MW

- **Providing the Monotonic Writes also involves two steps:**
  - In order for a server S to **accept a Write** at time t, the server's database, DB(S,t), must include the session's write-set.
  - Also, whenever a **Write is accepted** by a server, its assigned WID is added to the write-set.

Ensuring Session Guarantees (1)

- Table 1 summarizes for each guarantee what operation causes the session state to be updated and what operation requires checking this state to find a suitable server.

Table 1: Read/Write guarantees

Guarantee	session state updated on	session state checked on
Read Your Writes	Write	Read
Monotonic Reads	Read	Read
Writes Follow Reads	Read	Write
Monotonic Writes	Write	Write

Ensuring Session Guarantees (2)

- Table VII describes how the session guarantees can be met using a session object

Table VII. Implementation of Session Guarantees

Property	Session Updated:	Session Checked:
RYW	on write, expand write-set	on read, ensure write-set $\subseteq$ writes applied by site.
MR	on read, expand read-set	on read, ensure read-set $\subseteq$ writes applied by site.
WFR	on read, expand read-set	on write, ensure read-set $\subseteq$ writes applied by site.
MW	on write, expand write-set	on write, ensure write-set $\subseteq$ writes applied by site.

For example, to implement RYW, the system updates a user's session when the user submits a write operation. It ensures RYW by delaying a read operation until the user's write-set is a subset of what has been applied by the replica. Similarly, MR is ensured by delaying a read operation until the user's read-set is a subset of those applied by the replica.

Summary

- **Replication as a means to achieve high availability**
  - propagation and scheduling of operations
  - consistency
- **Example of sytems:**
  - Gossip
  - Bayou