Session Guarantees
Session Guarantees

- When client move around and connects to different replicas, strange things can happen
  - Updates you just made are missing
  - Database goes back in time
  - Etc.

- Design choice:
  - Insist on stricter consistency
  - Enforce some “session” guarantees

- Session
  - Abstraction of reads and writes
  - not atomic
  - allows abstraction of single, centralized server
  - can think of a sess. guarantee as subsetting acceptable servers
Terminology

- **Call write’s TS a “WID”,** assigned when server accepts write from client

- **DB(S,t)**: ordered seq. of writes seen by S at t

- Weak consistency: **DB(S1,t) ≠ DB(S2,t)**

- Eventual consistency: **DB(S1,t_∞) = DB(S2,t_∞)**

- session semantics “guaranteed”
  - they either hold, or app informed (WTF?)

*Non-commutative writes need to be applied in same order everywhere*
Supporting Session Guarantees

- Responsibility of “session manager”, not servers!
- WriteOrder(W1,W2) Boolean predicate

- Two sets:
  - read-set: set of writes that are relevant to session reads
  - write-set: set of writes performed in session

- Causal ordering of writes
  - Use scalar logical (Lamport) clocks
Read Your Writes

- Read operations reflect previous writes

RYW-guarantee:
- If Read R follows Write W in a session, and R is performed at server S at time t, then W is included in DB(S,t)

Examples:
- Changing password
- Deleting email
Monotonic Reads

- Successive reads reflect a non-decreasing set of writes

- A WS is complete for R, DB(S,t), if R returns same value whether against WS or DB(S,t)

- RelevantWrites(S,t,R)
  - returns the smallest set of Writes that is complete for Read R and DB(S,t)

- MR-guarantee:
  - If Read R1 occurs before R2 in a session and R1 accesses server S1 at time t1 and R2 accesses server S2 at time t2, then RelevantWrites(S1,t1,R1) is a subset of DB(S2,t2)

- Examples:
  - Calendar (appt’s blinking), mail (open msg after seeing a list of subjects)
WFR-guarantee:
- If Read R1 precedes Write W2 in a session and R1 is performed at server S1 at time t1, then, for any server S2, if W2 is in DB(S2) then
  - any W1 in RelevantWrites(S1,t1,R1) is also in DB(S2) and
  - WriteOrder(W1,W2).

Also affects users outside session

Bib example:
- correcting a bad item created by someone else
Writes Follow Reads cont.

- 2 constraints on Write operations
  - Constraint on eventual write order
  - Constraint on propagation

- Relaxed versions: WFRO, WFRP

- **Examples:**
  - Shared bibliographic database update: WFR
  - Bulletin board database: WFRP
    - ordering in DB doesn’t matter, as:
      - long as both there
      - reader associates replies w/ posting correctly
  - Shared bibliographic database write: WFRO
    - if writes are full bib items
    - don’t need earlier versions for display
Monotonic Writes

- New writes propagated after prior session writes

- MW-guarantee:
  - If Write W1 precedes Write W2 in a session, then, for any server S2, if W2 in DB(S2) then W1 is also in DB(S2) and WriteOrder(W1,W2).

- Also affects users outside session

- Example:
  - successive edits of a file, update of a library, followed by new app version that needs the update
Implementation

- **Only minor server cooperation:**
  - Return WID, set of relevant WIDs for a read, and all WIDs

- **Session manager must maintain for each session:**
  - Read-set: set of WIDs for the Writes that are relevant to session Reads
  - Write-set: set of WIDs for those writes performed in the session

- **Session manager part of client...**
Implementing Read Your Writes

- 2 basic steps:
  - Whenever a *new* Write is accepted by a server, its 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).
Implementing Monotonic Reads

● Similar 2 steps:
  - 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.
Implementing WFR and MW

- Require 2 additional server constraints
  - **C1**: New Writes are ordered after Writes that are already known to a server.
  - **C2**: Anti-entropy is performed such that if W2 is propagated from server S1 to server S2 at time t then any W1 in DB(S1,t) such that WriteOrder(W1,W2) is also propagated to S2.

- Most servers already have these properties
Implementing Writes Follow Reads

- 2 steps:
  - Each Read $R$ to server $S$ at time $t$ results in $\text{RelevantWrites}(S,t,R)$ 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 $\text{DB}(S,t)$. 
## Implementation Summary

<table>
<thead>
<tr>
<th>Guarantee</th>
<th>session state updated on</th>
<th>session state checked on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Write</td>
<td>Read</td>
</tr>
<tr>
<td>Read Your Writes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monotonic Reads</td>
<td>Read</td>
<td>Read</td>
</tr>
<tr>
<td>Writes Follow Reads</td>
<td>Read</td>
<td>Write</td>
</tr>
<tr>
<td>Monotonic Writes</td>
<td>Write</td>
<td>Write</td>
</tr>
</tbody>
</table>
Practical Implementation

- Session state (set of WIDs) may be large
- Use version vectors
  - Basically a vector clock, but 1 for each replica
- Can replace sets of WIDs with Version Vectors
Converting Write-Sets to Version Vectors

- **WS** $\rightarrow$ **Version Vector**
  - Set $V[S] = \text{the time of the latest WID assigned by server } S$
  - in the WS

- **WS1 $\cup$ WS2** $\rightarrow$ **Version Vector**
  - $V[S] = \max(V1[S], V2[S]) \ \forall S$

- **WS1 $\subseteq$ WS2**
  - Check $V2[i] \geq V1[i] \ \forall i$ (V2 dominates V1)

- **Hacks**
  - for relevant writes, most recent vector
Practical Implementation Cont.

```
Read(R,S) = {
    if MR then
        check S.vector dominates read-vector
    if RYW then
        check S.vector dominates write-vector
    [result, relevant-write-vector] := read R from S
    read-vector := MAX(read-vector,
                        relevant-write-vector)
    return result
}

Write(W,S) = {
    if WFR then
        check S.vector dominates read-vector
    if MW then
        check S.vector dominates write-vector
    wid := write W to S
    write-vector[S] := wid.clock
}
```

- Additional performance improvements:
  - “Latching” on to a server
  - Caching
Summary

- Very cool stuff!
  - abstraction of guarantees
    - idea
    - implementation

- How useful?
  - could be very useful.
View Consistency

“View consistency for optimistic replication”

by Ashvin Goel, Calton Pu, Gerald J. Popek
Idea

“Generalize session guarantees to apply to persistent as well as distributed entities.”

- Idea of using session guarantees on a file system:
  - guarantees on a single file, dirs
  - per client, or *group of clients*, possibly persistent
  - only server requirements is version info
  - implemented as stub on client
View Entries

● Each entity has view entry per file
  - used in each file access
  - optimism works (well, up to 8%...)

● Garbage collection
  - of view entries
    • toss VE when all replicas later
    • replica reconciliation, matrix time
  - of replica versions
Implementing Monotonic Writes

2 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.
- Whenever a Write is accepted by a server, its WID is added to the write-set.