Lecture 10: TM Pathologies

- Topics: scalable lazy implementation, paper on TM performance pathologies
Scalable Lazy Commit Algorithm

- Transactions that deal with different directories must proceed in parallel; transactions that deal with the same directory must necessarily be serialized.

- Before committing, a transaction must make sure there are no conflicts for its directories.

- Each directory has a token and a transaction can proceed with commit after acquiring every token for directories in its read and write sets.

- Tokens must be acquired in ascending order to avoid deadlocks; mechanisms are required to avoid starvation.
Considered Implementations

• LL: lazy versioning, lazy conflict detection, committing transaction wins conflicts

• EL: lazy versioning, eager conflict detection, requester succeeds and others abort

• EE: eager versioning, eager conflict detection, requester stalls
Pathology 1: Friendly Fire

- Two conflicting transactions that keep aborting each other
- Can do exponential back-off to handle livelock
- Fixable by doing requester stalls?

| VM: any |
| CD: eager |
| CR: requester wins |
Pathology 2: Starving Writer

- A writer has to wait for the reader to finish – but if more readers keep showing up, the writer is starved (note that the directory allows new readers to proceed by just adding them to the list of sharers)

- VM: any
- CD: eager
- CR: requester stalls
Pathology 3: Serialized Commit

- If there’s a single commit token, transaction commit is serialized
  - VM: lazy
  - CD: lazy
  - CR: any

- There are ways to alleviate this problem
Pathology 4: Futile Stall

- A transaction is stalling on another transaction that ultimately aborts and takes a while to reinstate old values.

- VM: any
- CD: eager
- CR: requester stalls
Pathology 5: Starving Elder

- Small successful transactions can keep aborting a large transaction
  - VM: lazy
  - CD: lazy
  - CR: committer wins

- The large transaction can eventually grab the token and not release it until after it commits
Pathology 6: Restart Convoy

- A number of similar (conflicting) transactions execute together – one wins, the others all abort – shortly, these transactions all return and repeat the process

- VM: lazy
- CD: lazy
- CR: committer wins
Pathology 7: Dueling Upgrades

- If two transactions both read the same object and then both decide to write it, a deadlock is created.
- Exacerbated by the Futile Stall pathology.
- Solution?

<table>
<thead>
<tr>
<th>VM</th>
<th>CD</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>eager</td>
<td>eager</td>
<td>requester stalls</td>
</tr>
</tbody>
</table>
Four Extensions

• Predictor: predict if the read will soon be followed by a write and acquire write permissions aggressively

• Hybrid: if a transaction believes it is a Starving Writer, it can force other readers to abort; for everything else, use requester stalls

• Timestamp: In the EL case, requester wins only if it is the older transaction (handles Friendly Fire pathology)

• Backoff: in the LL case, aborting transactions invoke exponential back-off to prevent convoy formation
Title

• Bullet