EECS 491
Introduction to Distributed Systems

Fall 2019

Harsha V. Madhyastha
Recap: Two-Phase Locking

- TC acquires locks on all necessary shards
- TC commits transaction if locks acquired, else aborts
- Disjoint transactions can execute concurrently
- Transactions with overlap run sequentially
- How to increase concurrency?
Managing Concurrency

- Two-phase locking is pessimistic
  - Acquire locks assuming conflict will occur

- Optimistic concurrency control
  - Execute transaction assuming no conflict
  - Verify that no conflict before commit
Optimistic Concurrency Control

- **Read** required data and compute results
  - TC ideally reads from local cache
- **Verify** that no conflicts
  - TC asks relevant partitions whether safe to commit
- **Commit** results
Safe Concurrency

- Say we have two transactions and T1 and T2
- What dependency in executing T1 and T2 concurrently without hurting serializability?

- Reason by identifying for each transaction
  - **Read set**: State that the transaction reads
  - **Write set**: State that the transaction updates
Safe Concurrency: Case 1

- T1 must commit writes before T2 begins read

When is this necessary?
  - If T1’s write set overlaps with T2’s read set
Safe Concurrency: Case 2

- Can run two transactions concurrently

- When is this feasible?
  - T1’s write set disjoint from union of T2’s read and write sets

- Example:
  - T1: $TotalSalary = \text{sum of employee salaries}$
  - T2: $MedianSalary = \text{median employee salary}$
Safe Concurrency: Case 3

- T1 and T2 can read in parallel, but T1 must complete commit before T2 begins commit

![Diagram showing T1 and T2 transactions]

- When is this necessary?
  - T1’s write set disjoint from T2’s read set, and T1’s and T2’s write sets overlap

- Example:
  - T1: TotalSalary = sum of faculty salaries
  - T2: TotalSalary = sum of graduate student salaries
Distributed Validation

- TC picks a timestamp based on local clock
  - Serial order is that imposed by timestamps

- TC includes timestamp in reads and writes

- When transaction T1 attempts to commit, invalidates reads of transaction T2 if:
  - T1 has lower timestamp than T2, and
  - Write set of T1 intersects with read set of T2
Distributed Validation

TC1

P1

P2

TC2

Read

Commit

Read
Announcements

- Submit peer feedback for Project 3
- Tomorrow’s discussion: Recap of material
- Next Tuesday’s lecture: Sample exam solutions
Mid-term exam

- 6:30 - 10:30pm next Thursday
  - No lecture that day
  - Will post room assignments on Piazza
  - Assigned seating

- Closed book; no notes
  - All questions will require analysis or construction
Exam strategies

- Take time to understand question
- Read every question carefully for assumptions about failures and delays

- Don’t get stuck in one question
  - Note recommended time

- Solution won’t require you to write tons of code
Preparing for exam

- Take the sample exam in an exam setting before Friday’s discussion
- Make sure you understand the rationale behind every design decision in projects
- Think about all “tools” you have learned and when each of them is useful
Logical clocks

- **Purpose:**
  - Associate every event with a logical timestamp
  - Order based on clock preserves “happens before”

- With both Lamport clock and vector clock,
  - If \( a \rightarrow b \), then \( C(a) < C(b) \)

- Only with vector clock,
  - If \( C(a) < C(b) \), then \( a \rightarrow b \)
Lamport clock

- Every process $P_i$ maintains a local clock $C_i$
  - Upon execution of each local event, set $C_i$ to $C_i + 1$
  - Upon receipt of message with clock value $t$, set $C_i$ to $\max(C_i + 1, t)$

- Use process ID to break ties
- $C_i(a).i < C_j(b).j$ when:
  - $C_i(a) < C_j(b)$, or $C_i(a) = C_j(b)$ and $i < j$
RSM with Lamport Clocks

C2 = 0.2
P1

C2 = 8.2
P2

C1 = 0.1

C1 = 3.1

C1 = 10.1

C2 = 8.2

C1 = 3.1

C1 = 10.1

October 31, 2019 EECS 491 – Lecture 17
RSM with Paxos

- Straightforward approach:
  - Client sends request to any replica
  - In each slot, execute Prepare and Accept phase

- Multi-Paxos:
  - Exploit spatial locality: a client will issue many reqs
  - Reuse leader across slots and skip Prepare phase

- Leader-based Paxos (also, Raft):
  - Pick a replica as leader; all clients contact leader
  - Since no contention, leader can skip Prepare