Welcome to ACS TA Session 6

Rodrigo, Julian, Leonardo, Li, Nikolaus, Tilman, Yijian

Department of Computer Science, University of Copenhagen

Academic year 2021-2022, Block 2

Agenda for today

Feedback on Theory Assignment 2

Feedback on Programming Assignment 3

Distributed Transactions

- The Two-Phase commit (2PC) protocol. Exercises

Theory Assignment 2 – Q1 Concurrency Control Concepts

- When answering questions like "*Show a schedule that is conflict-serializable, but could not be generated by a conservative two-phase locking scheduler*", some students show a schedule which contains the lock-unlock operations.
- However, we want you to write a schedule **without the locking operation**, and then analyze the possible locking method of your schedule.
- You can inject the lock-unlock operation to explain why your schedule is conflict serializable but could not be generated by C2PL.

Theory Assignment 2 – Q3 Recovery Concepts

3) Case 1:

When a transaction is committed (commit log record), the log tail is forced to stable storage, **even if we use no-force approach**. This is necessary because it guarantees the effect of this transaction is on the disk and will never be lost.

If a crash happens, we can recover the system by identifying which log records have been committed. Then, we can simply redo all the changes made by the transaction into main-memory pages.

Theory Assignment 2 – Q3 Recovery Concepts

3) Case 2:

Besides, when a page is stolen from memory to give space to other transactions operating upon a different page, every update log record that describes a change to this page is forced to disk. This is to safeguard that, in the presence of a crash, we can trace the operations of “loser” transactions and correctly UNDO them.

In conclusion, log forces (in these two situations) are sufficient for durability because it ensures that records of all changes to the disk / database is available / traceable.

Theory Assignment 2 – Q4 ARIES

- Only “*end*” transactions can be removed from the transaction table.
- If a transaction is committed, it wins and does not need *end* log record (if already found).
- For the REDO phase, the start LSN should be the smallest dirty *rec_LSN* in the Dirty Page table (we want to maintain the same order).
- For UNDO phase, the end LSN should be the oldest log record of failed transactions.
- The “end” record signifies that an abort/commit is completed. Committed transactions (i.e., those for which the commit record is part of the log on stable storage) are winners. The potential missing end record should be either written at the end of the REDO phase (page 116 in compendium).

Theory Assignment 2 – Q5 More ARIES

- One should consider all logs provided as durably stored (e.g., don’t need to create additional UNDO/REDO operations)
- Should have one CLR log record per operation issued by a failed transaction
- The end record must not be forgotten
- Pay attention to the correct order of UNDOing the operations of “loser” transactions

Programming Assignment 3 – (Code)

- Latency should be calculated for each worker thread individually.
    - Average them afterwards over the number of workers.
- Throughput should be calculated as described in the assignment text.
- The interaction methods should not call the bookstore API in a loop.
- AddCopies, AddBooks and BuyBooks all take sets, use it like such.
- Sorting the entire database on each interaction is expensive!

Programming Assignment 3 – (Q2: Plots, discussion)

- Read the questions properly
  - Lacking details of the name generator in the report
  - Explain your reasoning for opting for a parameter (e.g., why editorpick is always false for all records?)
  - Responses lacking preciseness:
    - “throughput for the RPC calls is better than the throughput for the local ones"
- “Explain the trends” part was missed by most.
- **Showing the graphs != Understanding the graphs**
- Missing RPC experiment (different address space)
  - At least write your expectations of how you think it would look
  - Better to show some understanding than give up

Programming Assignment 3 – (Q3: reliability, metrics)

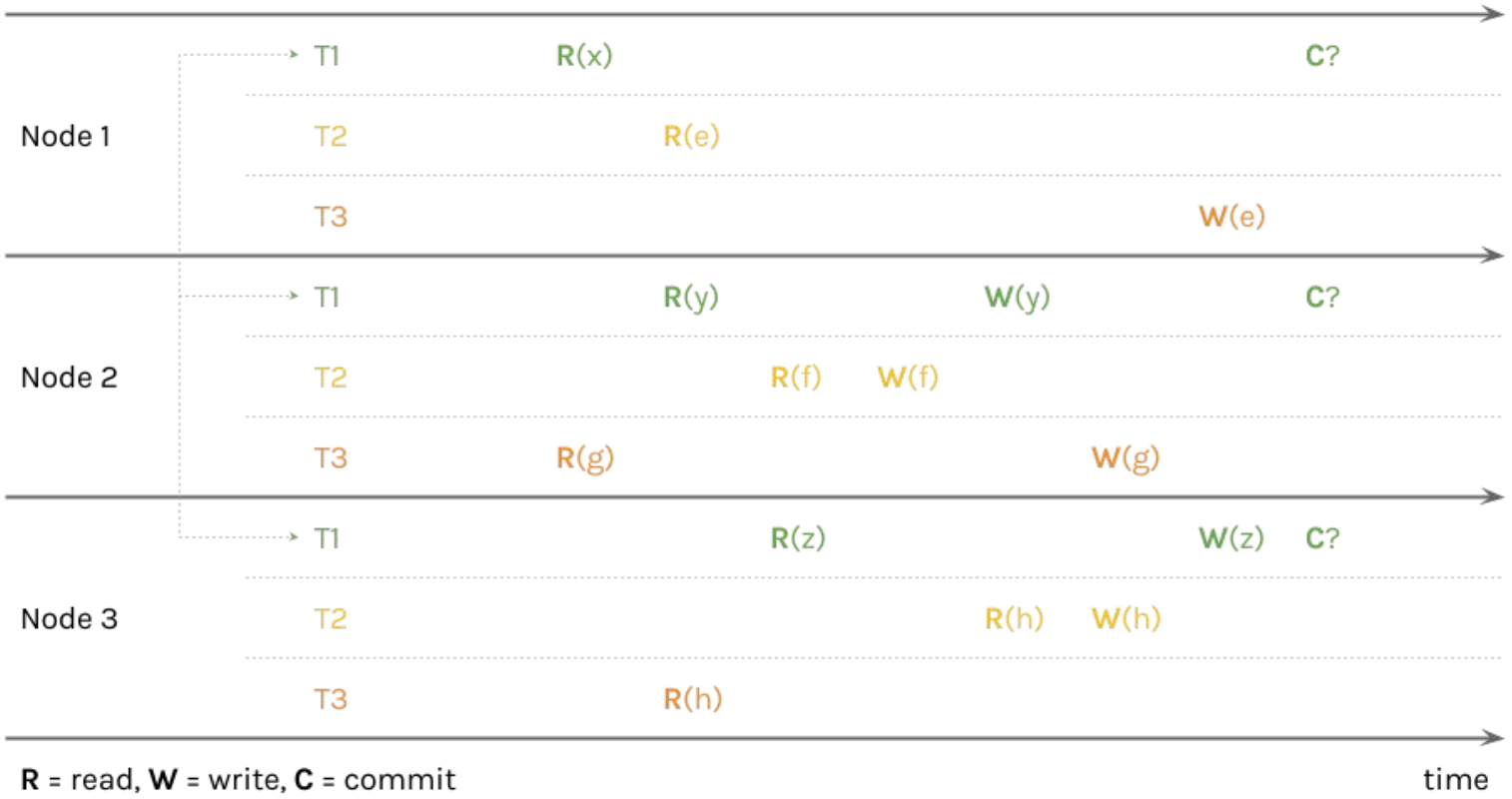
- A reliable metric reflects some invariant characteristics of the system. (Regardless of workload and setup)
- Please explain **WHY** the metrics you used are reliable.
  - What characteristics of the system does the metric show?
  - Will its behavior changes in different runs?
  - What different aspect to intend to capture with that?
- Metrics results can be different in a new setup, but performance have the same trend when doing the same tests for the same system in comparable setups.
- Successful Requests pr. time --> throughput. Not correctness.
- Fault-tolerance and scalability are not performance metrics.

Exercise 1: Distributed Transactions

For the distributed system shown in the next slide, answer the following questions:

1. What are the local wait-for graphs for the three nodes?
2. Is there any deadlock in the system?
3. What solutions the system nodes can employ to detect a deadlock? Explain in detail how one of the solutions is applied in the context of the system provided in this exercise.
4. At the end of the schedules, what transaction(s) will be able to issue a commit?

Exercise 1: Distributed Transactions



Exercise 2 – Linear vs Centralized 2PC

Consider a system that, instead of employing the traditional distributed commit 2PC protocol, the system employs a linear protocol for committing a distributed transaction.

A process P waits for a message from its left neighbor. If P receives YES and its own vote Is YES, then P forwards YES to its right neighbor. Otherwise, P forwards NO.

The **rightmost** process P_n will have all the information it needs to make a decision. If it receives a YES and its own vote is YES , then the decision is COMMIT, otherwise the decision is ABORT. After that, it forwards the decision to the left neighbor, and so on.

Each process that receives the decision message decides accordingly and then forwards that message to its left neighbor until the coordinator receives the message.

Figure 1. Graphical description of linear commit protocol

Exercise 2 - Questions

$$[P_{coord}] \rightarrow [P_1] \rightarrow \ldots \rightarrow [P_n]$$

1. If there are $n > 2$ participants, how many messages are sent when performing **centralized 2PC** compared to **linear 2PC**?
2. Assume a communication system with **unlimited** bandwidth. Assume further, that the processor time necessary to process a 2PC algorithm is 0 both for the coordinator and the participants. If message transmission time is t, how much time at least will it take when performing a **centralized 2PC** compared to a **linear 2PC**?
3. What is the outcome of the protocol if one process fails in the **linear 2PC**? *Hint: separate the problem into prepare and reply phase and before and after sending the message.*

Thank you

Rodrigo, Julian, Leonardo, Li, Nikolaus, Tilman, Yijian

Department of Computer Science, University of Copenhagen

Academic year 2021-2022, Block 2