
 CMU SCS

**Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications**


C. Faloutsos – A. Pavlo
Lecture#20: Overview of Transaction Management

 CMU SCS

Administrivia

- HW7 (Phase 1) is due **Tues April 1st**
- Recitations (always in SH 219):
 - Wed April 2nd 2:30-3:20
 - Wed April 9th 2:30-3:20

Faloutsos/Pavlo CMU SCS 15-415/615 2

 CMU SCS

Last Class

- Database Design
- Database Tuning

Faloutsos/Pavlo CMU SCS 15-415/615 3

CMU SCS

Today's Class

- Transactions Overview
- Concurrency Control
- Recovery

Faloutsos/Pavlo CMU SCS 15-415/615 4

CMU SCS

Motivation

Lost Updates
Concurrency Control → • We both change the same record ("Smith"); how to avoid race condition?

Durability
Recovery → • You transfer \$100 from savings → checking; power failure – what happens?

Faloutsos/Pavlo CMU SCS 15-415/615 5

CMU SCS

Motivation

Lost Updates
Concurrency Control → • We both change the same record ("Smith"); how to avoid race condition?

Durability
Recovery → • You transfer \$100 from savings → checking; power failure – what happens?

DBMSs automatically handle both issues: "transactions"

Faloutsos/Pavlo CMU SCS 15-415/615 6

CMU SCS

Concurrency Control & Recovery

- Valuable properties of DBMSs.
- Based on concept of transactions with **ACID** properties.
- Next lectures discuss these issues.

Faloutsos/Pavlo CMU SCS 15-415/615 7

CMU SCS

Transactions

- A **transaction** is the execution of a sequence of one or more operations (e.g., SQL queries) on a shared database to perform some higher-level function.
- It is the basic unit of change in a DBMS:
 - Partial transactions are not allowed!

Faloutsos/Pavlo CMU SCS 15-415/615 8

CMU SCS

Transaction Example

- *Move \$100 from Christos' bank account to his bookie's account.*
- Transaction:
 - Check whether Christos has \$100.
 - Deduct \$100 from his account.
 - Add \$100 to his bookie's account.

Faloutsos/Pavlo CMU SCS 15-415/615 9

CMU SCS

Strawman System

- Execute each txn one-by-one (i.e., *serial order*) as they arrive at the DBMS.
- One and only one txn can be running at the same time in the DBMS.

Faloutsos/Pavlo CMU SCS 15-415/615 10

CMU SCS

Problem Statement

- Better approach is to allow concurrent execution of independent transactions.
- **Q:** Why do we want that?
 - Utilization/throughput (“hide” waiting for I/Os)
 - Increased response times to users.
- But we also would like:
 - Correctness
 - Fairness

Faloutsos/Pavlo CMU SCS 15-415/615 11

CMU SCS

Transactions

- Hard to ensure correctness...
 - *What happens if Christos only has \$100 and tries to pay off two bookies at the same time?*
- Hard to execute quickly...
 - *What happens if Christos needs to pay off his gambling debts very quickly all at once?*

Faloutsos/Pavlo CMU SCS 15-415/615 12

CMU SCS

Problem Statement

- Arbitrary interleaving can lead to
 - Temporary inconsistency (ok, unavoidable)
 - “Permanent” inconsistency (bad!)
- Need formal correctness criteria.

Faloutsos/Pavlo CMU SCS 15-415/615 13

CMU SCS

Definitions

- A txn may carry out many operations on the data retrieved from the database
- However, the DBMS is only concerned about what data is read/written from/to the database.
 - Changes to the “outside world” are beyond the scope of the DBMS.

Faloutsos/Pavlo CMU SCS 15-415/615 14

CMU SCS

Formal Definitions

- **Database:** A fixed set of named data objects (A, B, C, \dots)
- **Transaction:** A sequence of read and write operations ($R(A), W(B), \dots$)
 - DBMS’s abstract view of a user program

Faloutsos/Pavlo CMU SCS 15-415/615 15

CMU SCS

Transactions in SQL

- A new txn starts with the **begin** command.
- The txn stops with either **commit** or **abort**:
 - If **commit**, all changes are saved.
 - If **abort**, all changes are undone so that it's like as if the txn never executed at all.

A txn can abort itself or the DBMS can abort it.

Faloutsos/Pavlo CMU SCS 15-415/615 16

CMU SCS

Correctness Criteria: ACID

- **Atomicity**: All actions in the txn happen, or none happen.
- **Consistency**: If each txn is consistent and the DB starts consistent, then it ends up consistent.
- **Isolation**: Execution of one txn is isolated from that of other txns.
- **Durability**: If a txn commits, its effects persist.

Faloutsos/Pavlo CMU SCS 15-415/615 17

CMU SCS

Correctness Criteria: ACID

- **Atomicity**: *“all or nothing”*
- **Consistency**: *“it looks correct to me”*
- **Isolation**: *“as if alone”*
- **Durability**: *“survive failures”*

Faloutsos/Pavlo CMU SCS 15-415/615 18

CMU SCS

Transaction Demo

Faloutsos/Pavlo CMU SCS 15-415/615 19

CMU SCS

Overview

- Problem definition & 'ACID'
- ➔ • **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability



Faloutsos/Pavlo CMU SCS 15-415/615 20

CMU SCS

Atomicity of Transactions A



- Two possible outcomes of executing a txn:
 - Txn might *commit* after completing all its actions.
 - or it could *abort* (or be aborted by the DBMS) after executing some actions.
- DBMS guarantees that txns are **atomic**.
 - From user's point of view: txn always either executes all its actions, or executes no actions at all.

Faloutsos/Pavlo CMU SCS 15-415/615 21

CMU SCS  **Mechanisms for Ensuring Atomicity** 



- We take \$100 out of Christos' account but then there is a power failure before we transfer it to his bookie.
- When the database comes back on-line, what should be the correct state of Christos' account?

Faloutsos/Pavlo CMU SCS 15-415/615 22

CMU SCS  **Mechanisms for Ensuring Atomicity** 



- One approach: **LOGGING**
 - DBMS *logs* all actions so that it can *undo* the actions of aborted transactions.
- Think of this like the black box in airplanes...

Faloutsos/Pavlo CMU SCS 15-415/615 23

CMU SCS  **Mechanisms for Ensuring Atomicity** 



- Logging used by all modern systems.
- **Q:** Why?

Faloutsos/Pavlo CMU SCS 15-415/615 24

CMU SCS  **Mechanisms for Ensuring Atomicity** 


- Logging used by all modern systems.
- **Q:** Why?
- **A:** Audit Trail & Efficiency Reasons
- *What other mechanism can you think of?*


Faloutsos/Pavlo CMU SCS 15-415/615 25

CMU SCS  **Mechanisms for Ensuring Atomicity** 



- Another approach: **SHADOW PAGING**
 - DBMS makes copies of pages and txns make changes to those copies. Only when the txn commits is the page made visible to others.
 - Originally from System R.
- Nobody actually does this...

Faloutsos/Pavlo CMU SCS 15-415/615 26

CMU SCS  **Overview**

- Problem definition & '**ACID**'
- **A**tomicity
-  • **C**onsistency
- **I**solation
- **D**urability



Faloutsos/Pavlo CMU SCS 15-415/615 27

CMU SCS  

Database Consistency

- **Database Consistency:** Data in the DBMS is accurate in modeling the real world and follows integrity constraints



Faloutsos/Pavlo CMU SCS 15-415/615 28

CMU SCS  

Transaction Consistency

- **Transaction Consistency:** if the database is consistent before the txn starts (running alone), it will be after also.
- Transaction consistency is the application's responsibility.
 - *We won't discuss this further...*

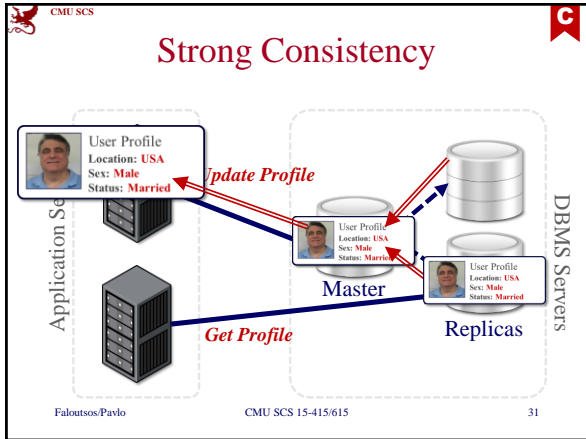
Faloutsos/Pavlo CMU SCS 15-415/615 29

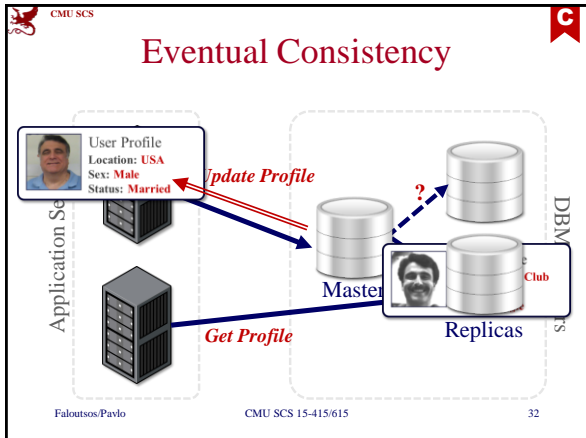
CMU SCS  

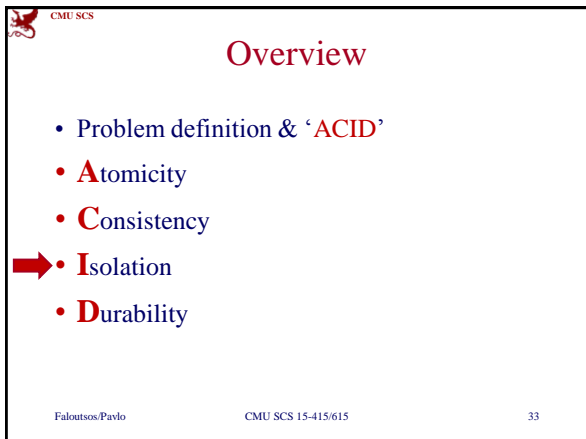
Strong vs. Weak Consistency

- In a distributed DBMS, the consistency level determines when other nodes see new data in the database:
 - **Strong:** Guaranteed to see all writes immediately, but txns are slower.
 - **Weak/Eventual:** Will see writes at some later point in time, but txns are faster.

Faloutsos/Pavlo CMU SCS 15-415/615 30







CMU SCS

Isolation of Transactions

- Users submit txns, and each txn executes *as if it was running by itself*.
- Concurrency is achieved by DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
- **Q:** How do we achieve this?

Faloutsos/Pavlo CMU SCS 15-415/615 34

CMU SCS

Isolation of Transactions

- **A:** Many methods - two main categories:
 - **Pessimistic** – Don't let problems arise in the first place.
 - **Optimistic** – Assume conflicts are rare, deal with them after they happen.

Faloutsos/Pavlo CMU SCS 15-415/615 35

CMU SCS

Example

T1	T2
BEGIN A=A+100 B=B-100 COMMIT	BEGIN A=A*1.06 B=B*1.06 COMMIT

- Consider two txns:
 - T1 transfers \$100 from B's account to A's
 - T2 credits both accounts with 6% interest.

Faloutsos/Pavlo CMU SCS 15-415/615 36

CMU SCS

Example

T1

BEGIN
A=A+100
B=B-100
COMMIT

T2

BEGIN
A=A*1.06
B=B*1.06
COMMIT

- Assume at first A and B each have \$1000.
- **Q:** What are the *legal outcomes* of running T1 and T2?

Faloutsos/Pavlo CMU SCS 15-415/615 37

CMU SCS

Example

- **Q:** What are the possible outcomes of running T1 and T2 together?
- **A:** Many! But A+B should be:
\$2000*1.06=\$2120
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. But, the net effect must be equivalent to these two transactions running **serially** in some order.

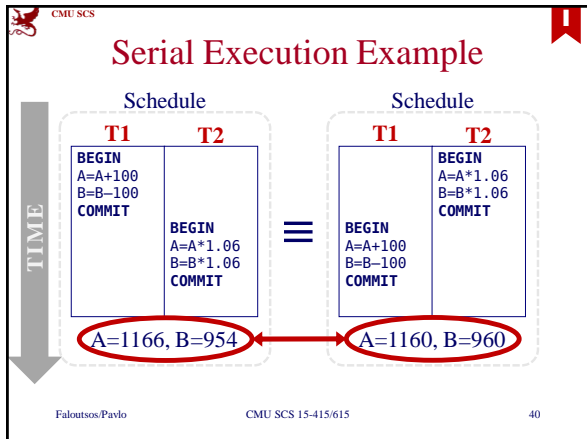
Faloutsos/Pavlo CMU SCS 15-415/615 38

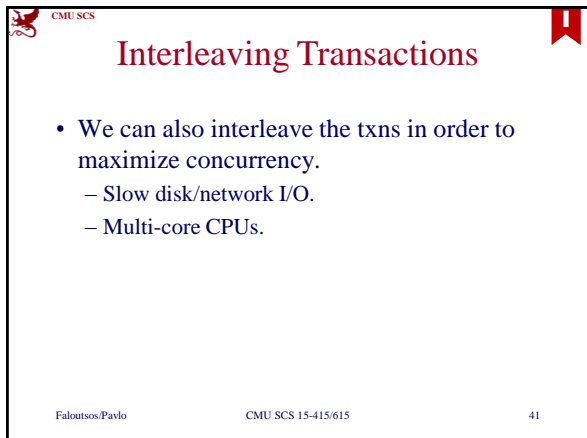
CMU SCS

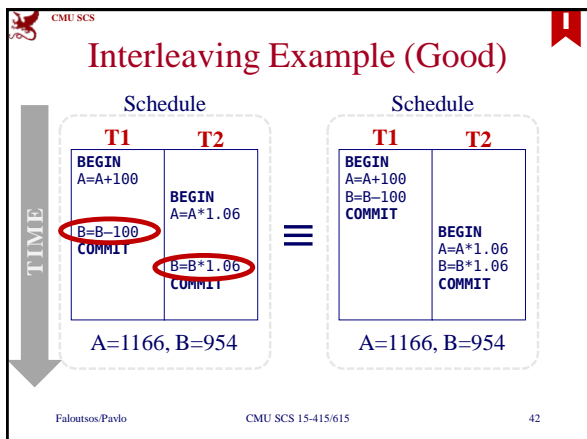
Example

- Legal outcomes:
 - A=1166, B=954 → **\$2120**
 - A=1160, B=960 → **\$2120**
- The outcome depends on whether T1 executes before T2 or vice versa.

Faloutsos/Pavlo CMU SCS 15-415/615 39







CMU SCS

Interleaving Example (Bad)

Schedule

T1	T2
BEGIN	BEGIN
A=A+100	A=A*1.06
	B=B*1.06
	COMMIT
B=B-100	
COMMIT	

\neq

 A=1166, B=954

 or

 A=1160, B=960

A=1166, B=960

The bank lost \$6!

Faloutsos/Pavlo CMU SCS 15-415/615 43

CMU SCS

Interleaving Example (Bad)

Schedule

T1	T2
BEGIN	BEGIN
A=A+100	A=A*1.06
	B=B*1.06
	COMMIT
B=B-100	
COMMIT	

A=1166, B=960

DBMS's View

T1	T2
BEGIN	BEGIN
R(A)	R(A)
W(A)	W(A)
	R(A)
	W(A)
	R(B)
	W(B)
R(B)	COMMIT
W(B)	
COMMIT	

Faloutsos/Pavlo CMU SCS 15-415/615 44

CMU SCS

Correctness

- **Q:** How do we judge that a schedule is correct?
- **A:** If it is *equivalent* to some *serial* execution

Faloutsos/Pavlo CMU SCS 15-415/615 45

CMU SCS

Formal Properties of Schedules

- **Serial Schedule:** A schedule that does not interleave the actions of different transactions.
- **Equivalent Schedules:** For any database state, the effect of executing the first schedule is identical to the effect of executing the second schedule.*

(* no matter what the arithmetic operations are!)

Faloutsos/Pavlo CMU SCS 15-415/615 46

CMU SCS

Formal Properties of Schedules

- **Serializable Schedule:** A schedule that is *equivalent* to some serial execution of the transactions.
- Note: If each transaction preserves consistency, every serializable schedule preserves consistency.

Faloutsos/Pavlo CMU SCS 15-415/615 47

CMU SCS

Formal Properties of Schedules

- **Serializability** is a less intuitive notion of correctness compared to txn initiation time or commit order, but it provides the DBMS with significant additional flexibility in scheduling operations.

Faloutsos/Pavlo CMU SCS 15-415/615 48

CMU SCS

Interleaved Execution Anomalies

- **Read-Write** conflicts (R-W)
- **Write-Read** conflicts (W-R)
- **Write-Write** conflicts (W-W)

- **Q:** Why not R-R conflicts?

Faloutsos/Pavlo CMU SCS 15-415/615 49

CMU SCS

Write-Read Conflicts

- Reading Uncommitted Data, “Dirty Reads”:

Faloutsos/Pavlo CMU SCS 15-415/615 50

CMU SCS

Read-Write Conflicts

- Unrepeatable Reads

Faloutsos/Pavlo CMU SCS 15-415/615 51

Write-Write Conflicts

- Overwriting Uncommitted Data

Faloutsos/Pavlo CMU SCS 15-415/615 52

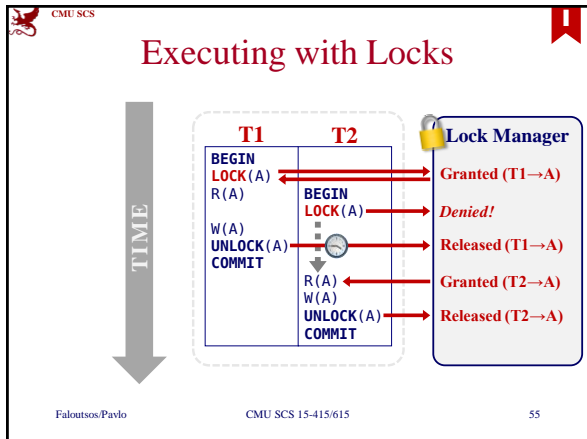
Solution

- Q:** How could you guarantee that all resulting schedules are correct (i.e., serializable)?
- A:** Use locks!

Faloutsos/Pavlo CMU SCS 15-415/615 53

Executing without Locks

Faloutsos/Pavlo CMU SCS 15-415/615 54



Executing with Locks

- **Q:** If a txn only needs to read 'A', should it still get a lock?
- **A:** Yes, but you can get a shared lock.

Faloutsos/Pavlo CMU SCS 15-415/615 56

Lock Types

- Basic Types:
 - **S-LOCK** – Shared Locks (reads)
 - **X-LOCK** – Exclusive Locks (writes)

	Shared	Exclusive
Shared	✓	✗
Exclusive	✗	✗

Faloutsos/Pavlo CMU SCS 15-415/615 57

CMU SCS

Executing with Locks

- Transactions request locks (or upgrades)
- Lock manager grants or blocks requests
- Transactions release locks
- Lock manager updates lock-table

- *But this is not enough...*

Faloutsos/Pavlo CMU SCS 15-415/615 58

CMU SCS

Executing with Locks

Faloutsos/Pavlo CMU SCS 15-415/615 59

CMU SCS

Concurrency Control

- We need to use a well-defined protocol that ensures that txns execute correctly.
- Two categories:
 - ➔ Two-Phase Locking (2PL)
 - Timestamp Ordering (T/O)

We will discuss T/O methods in future classes.

Faloutsos/Pavlo CMU SCS 15-415/615 60

CMU SCS

Two-Phase Locking

- **Phase 1: Growing**
 - Each txn requests the locks that it needs from the DBMS's lock manager.
 - The lock manager grants/denies lock requests.
- **Phase 2: Shrinking**
 - The txn is allowed to only release locks that it previously acquired. It cannot acquire new locks.

Faloutsos/Pavlo CMU SCS 15-415/615 61

CMU SCS

Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.

Transaction Lifetime

of Locks

Growing Phase Shrinking Phase

TIME

Faloutsos/Pavlo 62

CMU SCS

Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.

Transaction Lifetime

of Locks

Growing Phase Shrinking Phase

TIME

2PL Violation!

Faloutsos/Pavlo 63

CMU SCS

Strict Two-Phase Locking

- Txns hold all of their locks until commit.
- Good:
 - Avoids “dirty reads” etc
- Bad:
 - Limits concurrency even more
 - And still may lead to deadlocks

Faloutsos/Pavlo CMU SCS 15-415/615 67

CMU SCS

Strict Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.

Transaction Lifetime

of Locks

Growing Phase Shrinking Phase

TIME

Faloutsos/Pavlo 68

CMU SCS

Strict Two-Phase Locking

- Q: Why is avoiding “dirty reads” important?

T1 T2


BEGIN
R(A)
W(A)
R(B)
W(B)
ABORT

BEGIN
R(A)
W(A)
COMMIT

\$10
\$12

\$12


Faloutsos/Pavlo CMU SCS 15-415/615 69

CMU SCS 

Strict Two-Phase Locking

- **Q:** Why is avoiding “dirty reads” important?
- **A:** If a txn aborts, all actions must be undone. Any txn that read modified data must also be aborted.


Faloutsos/Pavlo CMU SCS 15-415/615 70

CMU SCS 


Locking in Practice

- You typically don’t set locks manually.
- Sometimes you will need to provide the DBMS with hints to help it to improve concurrency.
- Also useful for doing major changes.

Faloutsos/Pavlo CMU SCS 15-415/615 71

CMU SCS 

Overview

- Problem definition & ‘ACID’
- **A**tomicity
- **C**onsistency
- **I**solation
-  • **D**urability

Faloutsos/Pavlo CMU SCS 15-415/615 72

CMU SCS D

Transaction Durability

- Records are stored on disk.
- For updates, they are copied into memory and flushed back to disk at the discretion of the O.S.
 - Unless forced-output: **W(B)-fsync()**

This is slow!
Nobody does this!

Faloutsos/Pavlo CMU SCS 15-415/615 73

CMU SCS D

Transaction Durability

The diagram illustrates a transaction **T1** with operations **R(A)**, **W(A)**, and **COMMIT**. A **Buffer Pool** in **Memory** contains a page with **A=1**, which is linked to a corresponding page on the **Disk**.

Faloutsos/Pavlo CMU SCS 15-415/615 74

CMU SCS D

Transaction Durability

The diagram illustrates a transaction **T1** with operations **R(A)**, **W(A)**, and **COMMIT**. A **Buffer Pool** in **Memory** contains a page with **A=2**, which is not yet flushed to the **Disk** (which still has **A=1**). A lightning bolt indicates a crash.

Buffer is added to output queue but is not flushed immediately

Faloutsos/Pavlo CMU SCS 15-415/615 75

CMU SCS D

Write-Ahead Log

- Record the changes made to the database in a log *before* the change is made.
- Assume that the log is on stable storage.
- **Q:** What to replicate?
 - The complete page?
 - Single tuple?

Faloutsos/Pavlo CMU SCS 15-415/615 76

CMU SCS D

Write-Ahead Log

- Log record format:
 - **<txnId, objectId, beforeValue, afterValue>**
 - Each transaction writes a log record first, before doing the change
- When a txn finishes, the DBMS will:
 - Write a **<commit>** record on the log
 - Make sure that all log records are flushed before it returns an acknowledgement to application.

Faloutsos/Pavlo CMU SCS 15-415/615 77

CMU SCS D

Write-Ahead Log

- After a failure, DBMS “replays” the log:
 - Undo uncommitted transactions
 - Redo the committed ones

Faloutsos/Pavlo CMU SCS 15-415/615 78

CMU SCS D

Write-Ahead Log

T1

BEGIN

W(A)

W(B)

...

COMMIT

The DBMS hasn't flushed memory to disk at this point.

Safe to return result to application.

TxnId	ObjectId	Before Value	After Value
<T1	<A	100	200
<T1	<B	5	10
...			
CRASH!			

We have to redo T1!

Faloutsos/Pavlo CMU SCS 15-415/615 79

CMU SCS D

Write-Ahead Log

T1

BEGIN

W(A)

W(B)

...

COMMIT

```
<T1 begin>
<T1, A, 100, 200>
<T1, B, 5, 10>
:
CRASH!
```

We have to undo T1

Faloutsos/Pavlo CMU SCS 15-415/615 80

CMU SCS D

Recovering After a Crash

- At the end – all committed updates and only those updates are reflected in the database.
- Some care must be taken to handle the case of a crash occurring during the recovery process!

Faloutsos/Pavlo CMU SCS 15-415/615 81

CMU SCS

WAL Problems

- The log grows infinitely...
- We have to take checkpoints to reduce the amount of processing that we need to do.
- We will discuss this in further detail in upcoming classes.

Faloutsos/Pavlo CMU SCS 15-415/615 82

CMU SCS

ACID Properties

- **Atomicity:** All actions in the txn happen, or none happen.
- **Consistency:** If each txn is consistent, and the DB starts consistent, it ends up consistent.
- **Isolation:** Execution of one txn is isolated from that of other txns.
- **Durability:** If a txn commits, its effects persist.

Faloutsos/Pavlo CMU SCS 15-415/615 83

CMU SCS

Summary

- Concurrency control and recovery are among the most important functions provided by a DBMS.
- Concurrency control is automatic
 - System automatically inserts lock/unlock requests and schedules actions of different txns.
 - Ensures that resulting execution is equivalent to executing the txns one after the other in some order.

Faloutsos/Pavlo CMU SCS 15-415/615 84

CMU SCS

Summary

- Write-ahead logging (WAL) and the recovery protocol are used to:
 - Undo the actions of aborted transactions.
 - Restore the system to a consistent state after a crash.

Faloutsos/Pavlo CMU SCS 15-415/615 85

CMU SCS

Overview

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Recovery

Concurrency Control

Faloutsos/Pavlo CMU SCS 15-415/615 86
