Final Remarks on Concurrency Control
Experimental Design
Recovery: Basic Concepts

ACS, Marcos Vaz Salles

Do-it-yourself recap: Explain the difference!

- **Two-phase locking**
  - Conservative vs. strict schemes
- **Schedules and Serializability**
  - Conflict serializability vs. view serializability
- **Deadlocks**
  - Deadlock prevention vs. deadlock detection
- **Optimistic Concurrency Control**
  - Tests for WR conflicts vs. WW conflicts

Deadlock Prevention

- Assign priorities based on timestamps.
- **Lower timestamps get higher priority**, i.e., older transactions get prioritized
- Assume Ti wants a lock that Tj holds. Two policies are possible:
  - Wait-Die: It Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  - Wound-wait: If Ti has higher priority, Tj aborts; otherwise Ti waits

- If a transaction re-starts, make sure it has its original timestamp

Source: Ramakrishnan & Gehrke (partial)

From Rosenkrantz et al., TODS 3(2), 1978:

"In the WAIT-DIE system an older process is made to wait for younger ones and as it gets still older it tends to wait for more and more younger processes. **Although the older process will eventually terminate, it tends to slow down as it gets older.**

By contrast, in the WOUND-WAIT system, an older process never waits for a younger one except when the older process has wounded the younger and is waiting for the wound to take its effect. **The oldest process therefore runs roughshod through the system wounding any younger process in its path.** Thus, the older process get increased priority." (emphasis added)

Overheads in Optimistic CC

- Must record read/write activity in ReadSet and WriteSet per Xact.
  - Must create and destroy these sets as needed.
- Must check for conflicts during validation, and must make validated writes "global".
  - Critical section can reduce concurrency.
  - Scheme for making writes global can reduce clustering of objects.
- Optimistic CC restarts Xacts that fail validation.
  - Work done so far is wasted; requires clean-up.

- Still, optimistic techniques widely used in software transactional memory (STM), main-memory databases

Source: Ramakrishnan & Gehrke (partial)

Multiversion Concurrency Control (MVCC)

- This approach maintains a number of **versions** of a data item and allocates the **right version to a read operation** of a transaction. Thus unlike other mechanisms a **read operation in this mechanism is never rejected**.
- Side effect:
  - Significantly more storage (RAM and disk) is required to maintain multiple versions. To check unlimited growth of versions, a **garbage collection** is run when some criteria is satisfied
- Many commercial database systems implement a combination of MVCC and S2PL
- See compendium for more details

Source: Elmasri & Navathe, 2007 (partial)

Snapshot Isolation

- Often databases implement properties that are **weaker** than serializability
- **Snapshot isolation**
  - **Snapshots:** Transactions see snapshot as of beginning of their execution
  - **First Committer Wins:** Conflicting writes to same item lead to aborts
- May lead to **write skew**
  - Database must have at least one doctor on call
  - Two doctors on call concurrently examine snapshot and see exactly each other on call
  - Doctors update their own records to being on leave
    - No write-write conflicts: different records!
  - After commits, database has no doctors on call

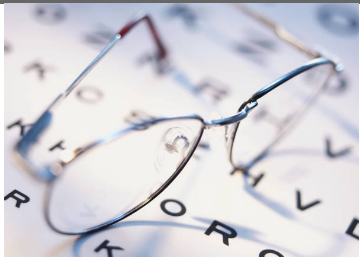
Transaction Support in SQL-92

- Each transaction has an access mode, a diagnostics size, and an isolation level.

Problem with SQL standard: snapshot isolation satisfies all requirements!

Isolation Level	Dirty Read	Unrepeatable Read	Phantom Problem
Read Uncommitted	Maybe	Maybe	Maybe
Read Committed	No	Maybe	Maybe
Repeatable Reads	No	No	Maybe
Serializable	No	No	No

Source: Ramakrishnan & Gehrke



What should we learn today?

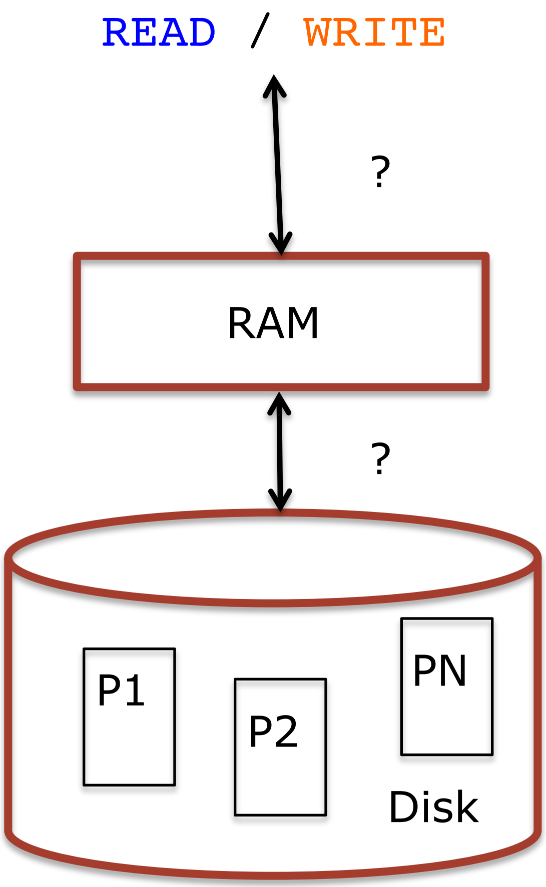
- Explain the three main methodologies for performance measurement and modeling: analytical modeling, simulation, and experimentation
- Design and execute experiments to measure the performance of a system
- Explain the concepts of volatile, nonvolatile, and stable storage as well as the main assumptions underlying database recovery
- Predict how force/no-force and steal/no-steal strategies for writes and buffer management influence the need for redo and undo
- Explain the notion of logging and the concept of write-ahead logging
- Predict what portions of the log and database are necessary for recovery based on the recovery equations

Techniques to Evaluate Performance

- Topic could be a whole course by itself! ☺
  - See refs in syllabus (books by Jain, Lilja)
- Three main techniques
  - Analytical Modeling
  - Simulation
  - Experimentation

Do-it-yourself Recap: Virtual Memory with Paging

- How could we build a two-level memory abstraction out of RAM and disk, with hopefully the latency of RAM and the size of disk?
- How did we handle READ and WRITE? What about page replacement?
- What were the guarantees we got in terms of atomicity and fault-tolerance?



Analytical Modeling

- Get intuition about system performance
  - Without actually implementing it!
- Remember our virtual memory system with paging? Simple model:

$$\begin{aligned} AverageLatency = {} & HitRatio * Latency_{Hit} + \\ & (1 - HitRatio) * Latency_{Miss} \end{aligned}$$

- With high hit ratio (say, >95%), average time can be pretty close to main memory
  - Some requests still require going to disk, of course, and take full disk latency blow
- How can we know the hit ratio?

Simulation

- Study properties of hard-to-model process, e.g., locality of workloads vs. hit ratio in cache

- Configure model with known parameters
  - In our example, $Latency_{Hit}$ and $Latency_{Miss}$

- Simulate behavior of system to get HitRatio

Simulation

- Pros
  - Effort may be smaller than full-blown implementation
  - Allows you to simulate "impossible" or hard-to-experiment-with scenarios → 10Ks of machines, next-generation flash disk not on the market yet

- Cons
  - Estimating parameters
  - Validating models and approximations
  - Choosing workloads

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Source: Dean

Choosing Workloads & Datasets

- Synthetic workloads & datasets
  - Example: Use Zipf distribution to generate workload of page accesses
- Real workloads & datasets
  - Example: Take **trace** of page requests from real application
  - Replay trace on your simulator
- Combinations also possible
  - Use real dataset but generate accesses using a distribution
- Issue: How can you tell if workload is representative?

Experimentation

- **Implement** real system or prototype
- **Measure** how it behaves with experiments
  - Most respected method
  - But also requires most effort
- **Profile** system to determine where time goes

Simple Factor Experimentation

- Understanding multiple influences
- Vary one factor at a time, keep others fixed

- Example: Skew of workload and size of cache
  - Skew=0.5, vary cache size from 1MB to 1GB
  - Cache size = 500MB, vary skew from 0 to 1

- Care required: Parameters may **influence** each other!

Benchmarking

- **Micro-benchmarks**
  - Measure a specific variable or piece of code, e.g., memory and disk latencies in small experiment to calibrate simulation model

- **Application-level benchmark**
  - Whole application designed to stress certain types of systems
  - **SPEC** benchmarks for compute-intensive apps, web servers, file systems, and many others
  - **TPC** benchmarks for databases

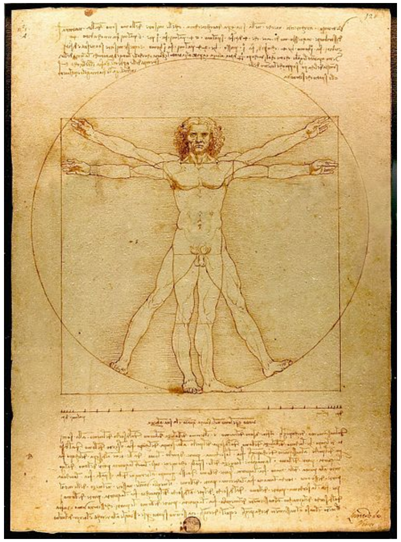
Designing a Micro-Benchmark

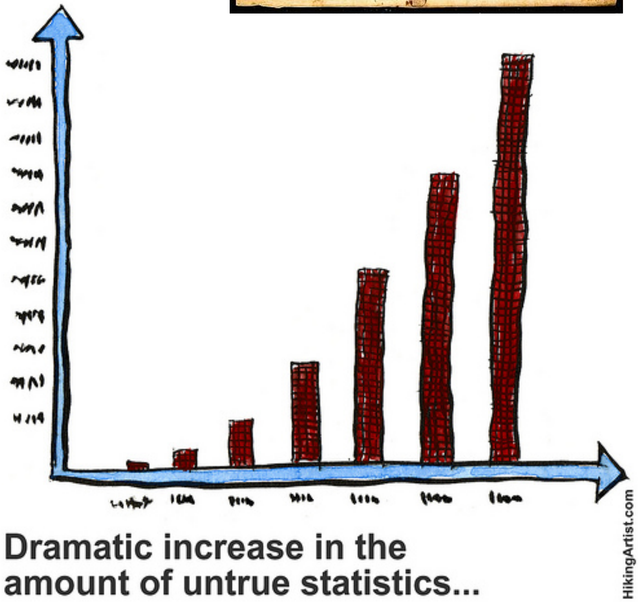
- How would you measure file scan performance of your filesystem?
- What performance metric would you measure?
- What are the factors that may affect performance?
- Which simple factor experiments would you use?

Hint: Think about your measurement program!

Necessary Care with Executing Experiments

- Select **event counts**
  - Number of pages/chunks read
  - Number of clock cycles elapsed → wall-clock time
- But control for **overhead** of event counting itself!
- **Sampling / monitoring**
  - e.g., I/O via iostat/vmstat
- "**Statistics** can prove anything?!" ☺
  - Number of measurements
  - Mean and variance
  - Confidence intervals
  - Dealing with outliers
  - Setup matters!





Dramatic increase in the amount of untrue statistics...

Comparing Alternatives

- Two systems, with throughput-oriented measurements R_2 and R_1
  - Both systems travel same distance D, i.e., do same work but take different time
  - $R_2 = D / T_2$; $R_1 = D / T_1$
- Speedup
  - $S_{2,1} = R_2 / R_1 = T_1 / T_2$
- Relative change
  - $\Delta_{2,1} = (R_2 - R_1) / R_1 = S_{2,1} - 1$
- **Example statements**
  - System 2 is 1.4 times faster than System 1
  - System 2 is 40% faster than System 1

Questions so far?

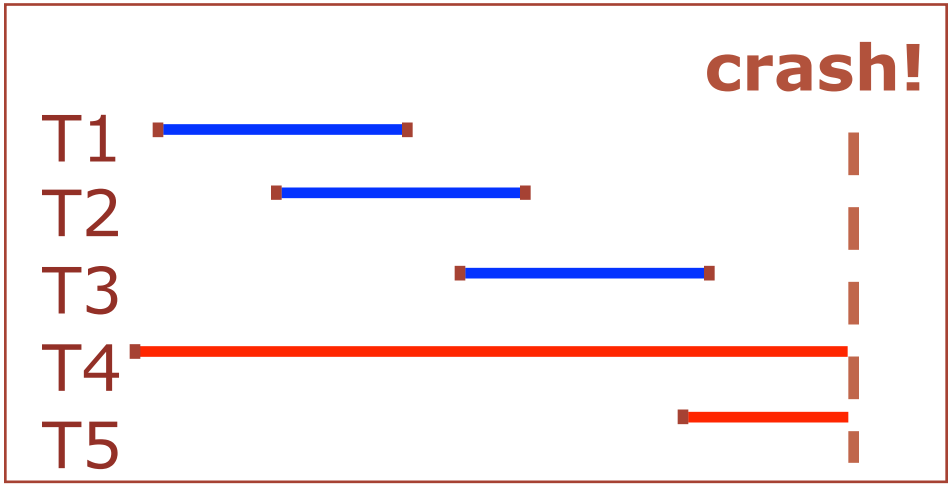
The many faces of atomicity

- **Atomicity** is strong modularity mechanism!
  - Hides that one high-level action is actually made of many sub-actions
- **Before-or-after** atomicity
  - == Isolation
  - Cannot have effects that would only arise by interleaving of parts of transactions
- **All-or-nothing** atomicity
  - == Atomicity (+ Durability)
  - Cannot have partially executed transactions
  - Once executed and confirmed, transaction effects are visible and not forgotten

Implementing All-or-Nothing Atomicity

- Atomicity
  - Transactions may abort (“Rollback”).
- Durability
  - What if system stops running? (Causes?)

- Desired Behavior after system restarts:
  - T1, T2 & T3 should be durable.
  - T4 & T5 should be aborted (effects not seen).



Source: Ramakrishnan & Gehrke (partial)

Assumptions

- Concurrency control is in effect
  - Strict 2PL, in particular
- Updates are happening "in place"
  - i.e. data is overwritten on (deleted from) memory using `READ` / `WRITE` interface.
  - We will use a two-level memory with buffer and disk
- Types of failures
  - Crash
  - Media failure
- Always fail-stop!

Source: Ramakrishnan & Gehrke (partial)

Volatile vs. Nonvolatile vs. Stable Storage

- Volatile Storage
  - Lost in the event of a crash
  - Example: main memory
- Nonvolatile Storage
  - Not lost on crash, but lost on media failure
  - Example: disk
- Stable Storage
  - Never lost (otherwise, that's it ☺)
  - How do you implement this one?

Source: Ramakrishnan & Gehrke (partial)

Surviving Crashes: How to handle the Buffer Pool?

Fill in the matrix:
When do you need to UNDO changes?
When do you need to REDO changes?

- Force every write to disk?
  - Poor response time.
  - But provides durability.
- Steal buffer-pool frames from uncommited Xacts?
  - If not, poor throughput.
  - If so, how can we ensure atomicity?

	No Steal	Steal
Force	Trivial (?)	
No Force		Desired

Source: Ramakrishnan & Gehrke (partial)

More on Steal and Force

- **STEAL** (why enforcing Atomicity is hard)
  - *To steal frame F:* Current page in F (say P) is written to disk; some Xact holds lock on P.
    - What if the Xact with the lock on P aborts?
    - Must remember the old value of P at steal time (to support UNDOing the write to page P).
- **NO FORCE** (why enforcing Durability is hard)
  - What if system crashes before a modified page is written to disk?
  - Write as little as possible, in a convenient place, at commit time, to support REDOing modifications.

Source: Ramakrishnan & Gehrke (partial)

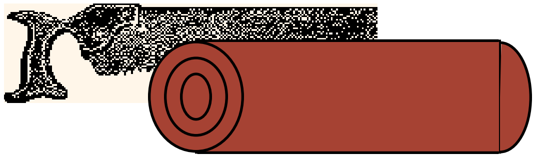
Undo/Redo vs. Force/Steal

	No Steal	Steal
Force	**No Redo** **No Undo**	**No Redo** **Undo**
No Force	**Redo** **No Undo**	$\boxed{\textbf{Redo} \; \textbf{Undo}}$

How do we support this option?

Source: Ramakrishnan & Gehrke (partial)

Basic Idea: Logging



- Record REDO and UNDO information, for every update, in a *log*.
  - Sequential writes to log (put it on a separate disk).
  - Minimal info (diff) written to log, so multiple updates fit in a single log page.
- Log: An ordered list of REDO/UNDO actions
  - Logical vs. Physical Logging
  - Example physical log record contains:

    <XID, pageID, offset, length, old data, new data>

  - Good compromise is physiological logging.

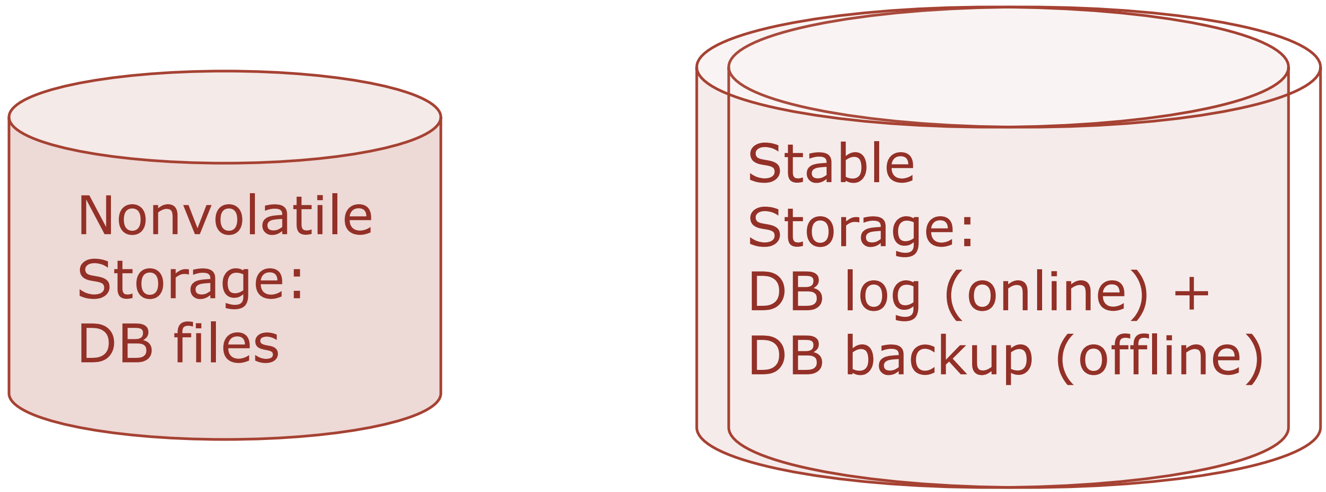
Source: Ramakrishnan & Gehrke (partial)

Write-Ahead Logging (WAL)

- Golden Rule: Never modify the only copy!
- The Write-Ahead Logging Protocol:
  1) Must force the log record for an update *before* the corresponding data page gets to disk.
  2) Must write all log records for a Xact *before commit*.
- #1 guarantees Atomicity.
- #2 guarantees Durability.
- Exactly how is logging (and recovery!) done?
  - We will study the ARIES algorithms.

Source: Ramakrishnan & Gehrke (partial)

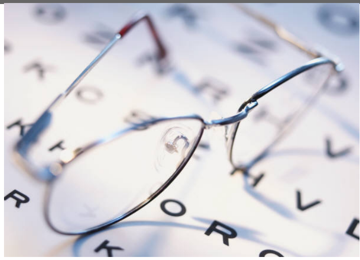
Recovery Equations



Discussion:

- Crash Recovery: volatile memory lost
  - Current DB = DB files + DB log → since when?
- Media Recovery: nonvolatile storage lost
  - Current DB = DB backup + DB log → since when?
- We will focus on crash recovery next

Source: Ramakrishnan & Gehrke (partial), Shasha & Bonnet (partial)



What should we learn today?

- Explain the three main methodologies for performance measurement and modeling: analytical modeling, simulation, and experimentation
- Design and execute experiments to measure the performance of a system
- Explain the concepts of volatile, nonvolatile, and stable storage as well as the main assumptions underlying database recovery
- Predict how force/no-force and steal/no-steal strategies for writes and buffer management influence the need for redo and undo
- Explain the notion of logging and the concept of write-ahead logging
- Predict what portions of the log and database are necessary for recovery based on the recovery equations