ADB Final 2022 -Theory

What are NoSQL Features?

1. The ability to horizontally scale "simple operations" throughput over many servers
2. the ability to replicate and to distribute data over many servers
3. A simple CLI or protocol (in contrast to SQLbinding)
    - Not bound to SQL, many simpler "query languages"
4. Weaker concurrency model than the ACID transactions
5. Can efficiently use distributed indexes and RAM for data storage
6. The ability to dynamically add new attributes to data records
    - No fixed schema

In summary, they are horizontal scaling, data replication, simple query languages, weak concurrency model, efficient use of RAM and indexes, flexible schema.

Can use the mnemonic HRSWEF

What is CAP and What is Important for NoSQL DBs?

- **Consistency:** do all clients see the same data?
- **Availability:** is the system always on?
- **Partition-tolerance:** even if communication is unreliable, does the system function?

CAP theorem states that we can only attain 2 of the 3. NoSQL chose AP as the most important, it sacrificed immediate consistency to get availability which is better than preventing clients from accessing data until eventually replicas get consistent.

How does the Recovery Manager (Undo/Redo) maintain durability and atomicity?

- Suppose a crash occurred
    - If a transaction wasn't fully complete its changes must be undone to maintain atomicity (0% or 100% done, no in-between)
        - Recovery manager in this case undoes uncommitted transactions with changes on the physical DB

    - If a transaction was fully complete its changes must be redone to maintain durability (changes by committed transactions must persist on disk)
        - Recovery manager in this case redoes committed transactions with changes (potentially) not on the physical DB

What is the Difference between C in CAP and ACID?

- Both refer to the word consistency but with different meanings in the general case
    - In CAP, consistency refers to data being the same across different replicas
    - In ACID, consistency refers to transactions not violating DB constraints

What are the Differences & Similarities between CAP and ACID?

- CAP
    - **Consistency:** do all clients see the same data?
    - **Availability:** is the system always on?
    - **Partition-tolerance:** even if communication is unreliable, does the system function?
- ACID
    - **Atomicity:** A transaction is either entirely performed or not performed at all
    - **Consistency:** A transaction does not violate any constraints on the database state
    - **Isolation:** A transaction should never interfere with others running concurrently
    - **Durability:** A transaction should have its changes persist on the DB after committing

Differences

- Definitions as above
- ACID applies to DBs using transactions (e.g. ,RDBMS)
- CAP applies to distributed systems regardless to whether they follow ACID or not

Similarities

- If the system follows ACID and C involves a constraint that data among replicas should be consistent then C becomes the same in both
- In this case, if the distributed system uses transactions, then it must choose CP over AP

What is WAL with an Example?

- Since we do in-place updates, we need a log to record transaction operations for recovery to be possible
- WAL means Write-ahead Logging and it refers to the concept that the log records for the data updates must be flushed to disk before the data updates (we must "write-ahead" the log records)
    - If this isn't true then if the crash occurs before update log records are flushed to disk but after the data itself is flushed to disk then after the crash, we have no log to know what needs to be undone or redone for such data.

Transaction:

1. Read(x)
2. x=x*2
3. Write(x)

This can't go to disk before this goes to disk

<START T>
<T,X,5,10>
<COMMIT T>

What is Trivial in no-steal/force and does it ensure D and A in ACID?

- NO-STEAL: If a transaction updates a buffer that buffer never goes to disk before the transaction commits
    - If a crash occurs the buffer will be gone
    - There is nothing on the disk to undo and the transaction is 0% complete
    - This satisfies atomicity and we never need to UNDO anything

- FORCE: Once a transaction commits any changes it made go to disk
    - If a crash occurs the buffers will be gone
    - There is nothing on the disk to redo and the changes by the transaction persist on the disk
    - This satisfies durability and we never have to REDO anything

- Thus, using NO-STEAL and FORCE means we don't need to do any recovery and atomicity, durability will be satisfied.

- The catch of course is that using NO-STEAL and FORCE means our DB is inefficient in terms of the number of buffers and IOs respectively.

What are NoSQL Databases' Different Choices in Concurrency Control?

- Locks
  - Field-level as in MongoDB
  - User-at-time locks also exist (allow one user to access at a time)
- Multi-version Concurrency Control
  - An alternative to locking as in multi-version timestamp ordering
  - Unlike the single-version case, allows much more concurrency (mainly reads)
- ACID
  - Pre-analyze transactions, follow same approaches as RBDMS
- None
  - Let multiple users access data in parallel
  - No guarantee on which version you read, may be okay.

Does 2PL Always Guarantee no Deadlocks? Prove it either case.

No, here is a proof by counterexample:
WL1(X) W1(X) WL2(Y) W2(Y) WL1(Y) WL2(X) UL1(X) UL1(Y) UL2(X) UL2(Y)

- System clearly follows 2PL for transactions 1 and 2
- But enters deadlock starting after timestep 6

What is the Type of 2PL with no Deadlocks?

- Conservative 2PL
- It avoids deadlocks by enforcing that any transaction should acquire all locks in the beginning, fully or none; thus, once a transaction stops waiting after starting it will never wait for any other transaction.

What is Strict 2PL and What is Special About it?

- A variant of 2PL that enforces every transaction to do all it's unlocks upon commit
- Guarantees a strict schedule
  - Have not covered what a strict schedule is but it guarantees that
    - Schedule is recoverable; once a transaction has committed, we never need to roll it back.
    - Schedule is cascadeless; when a transaction aborts that abort never cascades to other transactions
    - Schedule permits recovery by only undoing effect of aborted trans.

Why Can't we use 2PL with B+ Trees?

- 2PL is meant to prevent concurrent access to data items while maintaining serializability
- A B+ Tree is not a single data item and can be safely accessed concurrently (e.g., searching in left-subtree and deleting in a full right sub-tree of some node)
- It is possible to apply 2PL with B+ Trees but once we lock the root no other transaction can access it which would adversely limit performance

- Hence, we apply a different locking scheme that exploits how search and insertion/deletion work to lock only the necessary nodes. This involves alternating locks and unlocks (obvious for search) which violates 2PL but guarantees safe concurrent access of the B+ Tree

Is Phantom Effect when the Result of Query Changes from Time to Time?

- Yes, if the change is observed by the same transaction.
- More precisely, it happens when new record is being inserted by a transaction that satisfies a condition that the set of records accessed by another transaction must satisfy

Is FORCE Same as Immediate Update in that we Don't Need to REDO? 

- False.
- Recovery with Deferred Update is the REDO recovery we covered
- Recovery with Immediate Update is the UNDO/REDO recovery we covered
    - Immediate Update still doesn't guarantee that all updates are FORCED upon commit but unlike deferred update, it allows it.

Strict 2PL Prevents Dirty Read

- True
- Dirty read is the temporary update problem we covered
    - Transaction modifies a data item after reading the change by another failing transaction
    - Strict 2PL holds the locks until it commits or rollbacks
        - If at any point before commit, it aborts, the locks will be still held and no other transaction can make a dirty ready
            - Once transaction successfully aborted (rollback values) locks will be released