Name - PRABHAKAR KUMAR
Roll - IRM2017008.

MDM Assignment 2

Q1. The message and time complexity of Distributed 2PC is better than Linear 2PC. Why?

In Linear 2PC the coordinator sends a prepare message to all nodes, and each nodes response to the co-ord., and the coord. decides on response of all nodes. Then finally the coord. sends the commit or abort message. In distributed 2PC the coordinator sends a prepare message to the nodes, and the nodes send the responce to all other nodes and not only the coordinator.

For the message complexity, we can easily see that message complexity of distributed 2PC is greater than Linear 2PC since, each node is now has to send the response to all other nodes.

For Distributed :—

Suppose there are 'n' nodes.

Co-ord. Sends n prepare message
Each n nodes sends (n-1) response
∴ Messages = $n(n-1)+n$
$\approx O(n^2)$.

For Linear 2PC :-

Coord. Sends n prepare mess.
Each node sends 1 response to coord.
The coord. sends decision GA/GC message
Each node send the acknowledgement

∴ Messages = $O(n)$.

∴ Distributed 2PC = $O(n^2)$ messages
Linear 2PC = $O(n)$ messages.

In case of time complexity since the number of steps in the algorithm reduces from four majors to only 2 majors, hence the distributed 2PC has lower time complexity than Linear 2PC algorithm. Here we are assuming that the transmission time of messages b/w sites takes T time which is much larger than time taken for any decision computations.

Hence for Distributed 2PC
Time Complexity = 2T

Hence for Linear 2PC
Time Complexity = 4T.

Q2. Explain ACID property and discuss which ACID property can be safely removed without affecting the consistency preserving property of the execution of concurrent transactions.

A transactions is a single logical unit of work which access and possibly modifies the contents of a database.

Atomicity – The entire transaction takes place at once or doesn't happen at all.

Consistency – The ~~bat~~ database must be consistent before and after the transaction.

Isolation – Multiple Transactions occur independently without interference

Durability – The changes of a successful transaction occurs even if the system failure occurs.

Out of the four properties Atomicity, Consistency and Isolation are a must to maintain of consistent database. But even if a transaction is not durable then the changes made by it, would not persist and hence the state of database will have to roll back to the state before the transaction, which by virtue was also consistent. Hence Durability is the weakest of all four ACID properties, which even if not followed shall result in a consistent database.

Q3. Prove that if 2Phase policy is violated then it may not be possible to serialize the exec. of conc. transactions.

In 2Phase locking policy, a transaction before performing any operation on database, first acquires all locks on the required data items, in the growing phase. This ensures that all the data items are in a consistent state. If this is not follow then it may occur that

the data items are not in a consistent state.

Ex:- let A = 10, B = 20, C = 30, D = 40.

T1 :-	T2 :-
R(A)	R(C)
R(B)	W(C = C*10)
W(A = A+B)	
R(C)	
W(C = C+A)	

If T1 → T2, then

A = 30, B = 20, C = 600, D = 40.

But for a non serial sequence not following the 2Phase locking protocol. as :-

T1. L(A)
T1. R(A)
T1. L(B)
T1. R(B)
T1. W(A = A+B)
T2. L(C)
T2. R(C)
T2. W(C = C+10)
T2. Rel(C)
T1. L(C)
T1. R(C)
T1. W(C = C+A)
T1. Rel(A)
T2. Rel(B)
T1. Rel(C)

This non serial sequence results in

A = 30
B = 20
C = 330
D = 40

which is not consistent with the serialized result.

Q4. Consider developing your own concurrency control mechanism based on dynamic attributes of transactions.

We may define an attribute associated with every transaction as :—

State = 1 ⇒ Growing phase
State = 2 ⇒ Execution
State = 3 ⇒ Shrinking phase.

In the locking mechanism we propose a mechanism similar to Strict 2 phase locking protocol. Each site has a request list very much like a queue but where it can iterated over the elements and not only access the top most element. For a transaction to acquire a locks :-

1) It has an array of sites to be locked, initialised to zero.
2) It sends a Req(site) message to all sites
3) If it receives granted site message from all sites it proceed to state 2.
4) If any site sends a wait message it sends Release Site(0) message to all sites from whom it had got a granted message.

5) If waits until any of the site now sends a Granted Site S_i message, upon which it resumes from order 2, but ignores S_i until only setep 2.

6) Upon completion send Release Site(1) to all.

At data sites :-

1) Upon Receiving Req T_i from i^{th} transaction; append T_i to the end of list.

2) Send Granted Site to the transaction on top. say T_j.

3) If T_j responds with Unlock Site(0), iterated over to the next transaction in the list.

4) If T_j responds with Unlock Site(1), remove the T_j from top of list and resume from Step 1, with the new head of list.

Note -

Unlock Site (0) → Implies transaction was not able to acquire all locks, hence is releasing the lock, but still needs it to execute.

Unlock Site(1) → Implies transaction has completed and no more need the data item.

For the commit protocol we propose that :- T_i

1) If a transaction wants to commit, it sends a commit message to all others.
2) If all other sites have transaction with phase 1 or state 1, or no transaction, then they reply with OK message, O/W a wait message
3) If T_i receives OK from all sites, it commits and sent commit OK message to all sites to which they commit.
4) If T_i receives a wait message then it waits till the site sends an OK message.
5) Once a site has sent an OK message, it then needs to sit idle and can not either acquire any lock or can executed any more execution.

Q5 Consider Bayou approach ... ... handling conflicts.

Bayou approach for database manipulation in weak connectivity environment, works upon session instead of support. It is built on peer to peer architecture with a number of replicated servers weakly connected to each other.

Bayou systems provides dependency checks for automatic conflict detection and merges procedures for resolution. Session here is an abstraction for a sequence of read & write operations performed during the execution of an application.

Dependency Checks :-

- Application supplied query & expected result.
- Query is run at the server against against its current data.
- If check fails, invoke merge procedure.

Pseudo Code for write -

```
Bayou_Write(update, dep_check, mergeproc)
{
```

query

if (db_eval(dep-check.) is not same as
dep-check.expected_result)
resolved_update = Interpret(merge_prx);
else
resolved_update = update.
DB_apply (resolved_update);
}

Merge Procedures

- High-level program with application-spec. knowl.
- Run by the Server
- Performed Atomically as part of Writes
- Attempt to Resolve the Conflict.
- Produce a Revised update to apply.

Pseudo code for Write :-

1) run update query and get the o/p.
2) check for dependency query and store it as a Condition say C.
3) Execution of merge process begins
4) First get a list of alternative queries that can be executed in order if the original query is not satisfied.

5) First check for validation of original query, if true then —
newupdate = original query
and then return newupdate.

6) For each alternate updates, loop over and perform :—
(1) Check if alternate update i creates a valid result as per the condition C.
(2) If successful then
newupdate = alternate query i
and break
(3) If not successful then
continue iteration.

7) If newupdate is empty, then
newupdate = Error message

8) return newupdate.