Data Write Conflict summary

* Conflict resolution/prevention techniques for Multi-Master: -
* Replication between 2 masters 🡪 Conflict occurs only during asynchronous replication. In synchronous replication changes are done using 2PC or 3PC. Each one will maintain previous and current updated time stamp of each record. These details are shared between masters along with the data change. Using time stamp it can detect if there is write conflict. This logic work if primary key is associated with each record. Refer to conflict resolution in Volt-DB  
  <https://docs.voltdb.com/UsingVoltDB/DbRepHowToActive.php#DbRepConflictProcess>

| **Action** | **Possible Conflict** | **Result for Tables with Primary Key** |
| --- | --- | --- |
| INSERT | Constraint violation | Rejected |
| UPDATE | Missing row Timestamp mismatch Constraint violation | Rejected Last transaction wins Rejected |
| DELETE | Missing row Timestamp mismatch | Accepted (no op) Accepted |

* Multi-master conflict can also be prevented by ensuring write on a table/schema is happening, only at one place.
* Let say, there are 3 tables T1, T2, T3 and 3 masters M1, M2, M3
* M1 will allow write on T1, while read from T2 & T3
* M2 will allow write on T2, while read from T1 & T3
* M3 will allow write on T3, while read from T1 & T2
* Based on the above configuration, if write for T1 goes to M2 or M3, they will re-direct that request to M1
* This way write will happen at only one place while read can happen from any masters. Refer to Multi-Master Conflict Prevention doc
* e.g. write request for USA customer will go to Datacenter in USA, for Europe customer will go to Europe datacenter. If Europe datacenter receive write from USA customer, it will re-direct the write request to USA datacenter
* GALERA Conflict Resolution
* Each write at the node will be collected in a write set
* The group communication will broadcast the write set to all the node
* GALERA can do conflict detection between different write sets, so enqueued (but not yet committed) write sets are protected from local conflicting commits until our replicated write set is committed
* When the write set is applied on a given node, any locking conflicts it detects with open (not-yet-committed) transactions on that node causes that open transaction to get rolled back
* Write sets being applied by replication threads always win
* The implementation of certification-based replication in GALERA Cluster depends on the global ordering of transactions. GALERA Cluster assigns each transaction a global ordinal sequence number, or seq\_no, during replication. When a (new) transaction reaches the commit point, the node checks the sequence number against that of the last successful transaction. The interval between the two is the area of concern, given that transactions that occur within this interval have not seen the effects of each other. All transactions in this interval (are still waiting for being applied at the current node, they are still not committed) are checked for primary key conflicts with the transaction in question. The certification tests fail if it detects a conflict
* The enqueue transaction (which has passed certification test) will be committed at each node asynchronously
* Manually resolve 🡪 store both the value and let user decide which one should be kept and which one to be rejected. This is same as Merge tool we use in Perforce or any version control system
* Trigger store procedure 🡪 there are tools like Bucardo & BDR, which allow code to be written that will be trigger when write conflict is detected. Both the values will be passed to the stored procedure, user can write the logic to resolve it
* ensuring that all transactions for a single user session, or associated with a purchase order, are directed to the same cluster

=====================================================================================

**For avoiding write-write conflict**

* Use Qurom based protocol W > N/2, to avoid write-write conflict
* Database Sharding 🡪 user profile is present in separate database. All write is sent to that data base only. All the update operations for a user is send to a specific database i.e. sticky session
* Use time-stamp (Lamport or Vector) and perform all the write operation in that order after checking if it’s not violating any constraint.   
  E.g. Two users trying to create profile with same user id at two different geographical separated regions.
* Before performing any write operation, the program should validate if user id is not existing in the system.
* Write conflict can be avoided by using **strict consistency** for user registration operation
* This can be achieved by using sequential consistency with time stamp