MapReduce and Transactions

Module Objectives

In this module, you will look at:

- MapReduce, DataSets and Transactions
- Key difference of MapReduce in Transactions
- Failure of a MapReduce Job

Transactions in MapReduce

For a MapReduce job that interacts with DataSets, the system creates a long-running transaction

Similar to the transaction of a Flowlet or a Procedure, here are rules to follow:

- Reads can only see the writes of other transactions that were committed at the time the long-running transaction was started
- All writes of the long-running transaction are committed atomically, and only become visible to others after they are committed
- The long-running transaction can read its own writes

Key Difference of MapReduce in Transactions

However, there is a key difference between Flowlets and Procedures: **long-running transactions do not participate in conflict detection**

- If another transaction overlaps with the long-running transaction and writes to the same row, it will not cause a conflict but simply overwrite it
- It is not efficient to fail the long-running job based on a single conflict
- Because of this, it is not recommended to write to the same DataSet from both real-time and MapReduce programs
- It is better to use different DataSets, or at least ensure that the real-time processing writes to a disjoint set of columns

---

What Happens If A Job Fails?

Important:

MapReduce framework will reattempt a task (Mapper or Reducer) if it fails

If the task is writing to a DataSet, the reattempt of the task will most likely repeat the writes that were already performed in the failed attempt

Highly advisable that all writes performed by MapReduce programs be idempotent

---

Module Summary

You should now understand:

- MapReduce, DataSets and Transactions
- Key difference of MapReduce in Transactions
- Failure of a MapReduce Job

Module Completed