Updates & View Maintenance in Soft Real-Time Database Systems

Reynold Cheng
Department of CSIS
The University of Hong Kong
Joint work with Ben Kao, K.Y. Lam, Brad Adelberg and Tony Lee

Contents
- Problem Definition
- Temporal Correctness
- Transaction Scheduling
- Future Work
Real-Time Database Systems

- A Real-Time Database System (RTDB) is a database system, with transactions associated with time constraints (deadlines).
- Application:
  - In a financial database system, stocks and financial instruments are monitored.
  - If an opportunity is discovered, the system will perform a trading transaction before a deadline.
  - If the trading transaction cannot be finished before a deadline, then it will be useless.

Conceptual Model of RTDB
Timing Requirements of Application Transactions

- **Transaction Timeliness**: How fast the system responds to an application transaction request
  - Can a transaction complete before its deadline?

- **Data Timeliness**: How fresh the data is
  - A transaction should read fresh data
  - Stale data is less useful to a transaction

The 2 Requirements Conflict!

- To keep the database fresh, updates and recomputations must be applied promptly (to satisfy *data timeliness*)
- These are extremely heavy loads
  - 500 updates per second in the U.S. market
  - Some recomputations are complex and long
- Application transactions get little share of system resources (this hurts *transaction timeliness*)
Problems to be Studied

- How to satisfy these both timing requirements?
  - By redefining the meaning of “data timeliness”
- Design scheduling algorithms to handle updates, recomputations and application transactions efficiently.

Temporal Correctness

- If data are always kept fresh, transactions miss deadlines easily.
- Can we relax the data timeliness constraint?
- **Instantaneous system:** An ideal system which applies updates/recoms as soon as they arrive, taking zero time to do it.
- A data item is absolutely consistent (AC) if it timely reflects the state of an external object.
- A set of data items is relatively consistent (RC) if their values reflect the states of the external objects at the same time instant.
AC and RC: An Example

\[ A \quad A_1 \quad B_1 \quad B \quad B_2 \quad A_2 \]

- \( A_1 \) and \( B_2 \) are AC pairs.
- \( A_1 \) and \( B_1 \) are RC but not AC pairs.
- \( A_2 \) and \( B_1 \) are NOT consistent pairs.

ACS and RCS

- In practice, an ideal system does not exist. So we need to define real systems.
- In an ACS (Absolute Consistent System), all transactions must read AC data \( w.r.t \) commit time. No transactions can read old or inconsistent data.
- We can relax the requirement of data freshness by allowing transactions to read slightly old data.
- In an RCS (Relative Consistent System), a transaction can read RC data, which are not older than transaction's start time - \( \Delta \).
ACS vs RCS: An Example

- Readset of ACS: \( (A_2, B_1) \)
- Readset of RCS: \( (A_1, B_1), (A_2, B_1) \)

Advantages of RCS over ACS

- In an ACS, if an incoming update/recom conflicts with a transaction, it has to be aborted.
- In an RCS, once a transaction successfully reads all the items it needs, it will not be aborted by later updates due to data conflicts.
- Hence transactions have a better chance of finishing before its deadline.
- In an RCS, freshness is compromised in return for fewer transaction aborts.
Schedulers for ACS and RCS

- To meet different levels of consistency requirements, we need scheduling policies for updates, recomputations and application transactions:
  - **URT** *(Updates & Recomputations first, Transactions later)*
    - A simple but poor method because application transactions only get a small share of system resources
  - **OD** *(On Demand)*
    - Incoming updates/recomputations are NOT executed until the data items are *required* by transactions
    - Significantly reduces no. of updates/recomputations and hence provide *more* system resources for transactions

Implementing the Schedulers

- Implementing a scheduling policy for a system is non-trivial.
- A direct application of URT/OD does not meet ACS /RCS requirement. We also need concurrency control.
- The ACS schedulers are based on *HP-2PL*.
- The RCS schedulers are based on a *multi-version database*. 
Future Work

- We now study the version selection problem in an RCS.
- The simulation model can be modified so that the readset of a transaction is known before its execution. This may have a profound effect on RCS.
- The simulation model can be expanded from single processor and single disk, to multiple processors and multiple disks.

Logical Correctness

- Serial Schedule
- Serializable Schedule
- Concurrency Control Protocols e.g. 2PL, OCC

Temporal Correctness

- Instantaneous System
- ACS, RCS
- Scheduling Policies e.g. URT, OD