Timestamping

A timestamp is a unique identifier created by the DBMS to identify a transaction.

Sequentially assigned by the system

We will use TS(T) to identify transaction T.

we associate timestamps with each database item X:

`read_TS`: the largest timestamp of those transactions that have read `X`

`write_TS`: the largest timestamp of those transactions that have written `X`

We can use these and timestamps on transactions to enforce a `timestamp ordering', which will guarantee serializability.

For every request by a transaction `T` to read an item (`read_item(X)`) or to write an item (`write_item(X)`), must check timestamp of transaction with timestamps of database item

If ordering is violated, operation is rejected

T issues a write_item(X)

```
if (read_TS(X) > TS(T)) or
   (write_TS(X) > TS(T))
     rollback(T)
else
    allow operation
    write_TS(X) := TS(T)
```

T issues a `read_item(X)`:

```
if write_TS(X) > TS(T)
    rollback T
if write_TS(X) =< TS(T)
   allow read
   read_TS(X) := max (TS(T), read_TS(X))
```

Time-stamping can be used to enforce conflict serializable schedules:

Consider:

```
T1                          T2
read_item(X);
 X := X+M;
                             read_item(X);
                                    X := X-N;
write_item(X);
                             write_item(X);
```

With time-stamping:

Let `TS(T1)` = 1; Let `TS(T2)` = 2;

```
   T1              T2          r_ts(x)  w_ts(x)
read_item(X);                  1
 X := X+M;
             read_item(X);     2
                X := X-N;

write_item(X);
/* rollback */
              write_item(X);                2
```

Thomas' write rule

A common modification to the timestamping ordering is as follows:

```
write_item(X):

if (read_TS(X) > TS(T))
     rollback(T)
else if write_TS(X) > TS(T)
    ignore write and allow T to continue
else
    allow operation
    write_TS(X) := TS(T)
```

Effect: no longer guarantees conflict serializability but rejects fewer operations.

One of the disadvantages of time-stamping is that a transaction maybe repeatedly aborted and restarted (cyclic restart problem). Similar to unfairness in a locking scheme

Although 2PL and time-stamping both guarantee conflict-serializable schedules, not all schedules allowed under 2PL will be allowed under time-stamping and vice-versa

Multi-version time-stamping:

Time-stamping can be extended to allow multi-version techniques

For each database item `X`, we maintain a set of versions `X1 ... XN`

If a read or write request is submitted, the timestamp of the transaction can be checked against those of the items involved in the read or write, and the appropriate version used.

Limits the number of transactions that need to be restarted and hence leads to increased concurrency and increased throughput.

Requires much more storage

Granularity

For both locking and time-stamping we have assumed that all lock and timestamps were maintained for database items. The database item could be a record, a field value, a block, a whole file or the entire database.

The larger the item the lower the degree of concurrency.

The smaller the item, the higher the number of locks (or timestamp values) that have to be maintained.

Although 2PL and time-stamping both guarantee conflict-serializable schedules, not all schedules allowed under 2PL will be allowed under time-stamping and vice-versa