Distributed STMs

- STMs are being employed in new scenarios:
  - Database caches in three-tier web apps (FénixEDU)
  - HPC programming language (X10)
  - In-memory cloud data grids (Coherence, Infinispan)

- New challenges:
  - Scalability
  - Fault-tolerance
Partial Replication

- Each site stores a partial copy of the data.

- **Genuine** partial replication schemes maximize scalability by ensuring that:
  - Only data sites that replicate data item read or written by a transaction \( T \), exchange messages for executing/committing \( T \).

- Existing 1-Copy Serializable implementations enforce distributed validation of read-only transactions [SRDS10]:
  - considerable overheads in typical workloads
Issues with Partial Replication

- Extending existing local multiversion (MV) STMs is not enough.
- Local MV STMs rely on a single global counter to track version advancement.

Problem:
- Commit of transactions should involve ALL NODES

NO GENUINENESS = POOR SCALABILITY
In the execution/commit phase of a transaction $T$, ONLY nodes which store data items accessed by $T$ are involved.

- It uses multiple versions for each data item.
- It builds visible snapshots = freshest consistent snapshots taking into account:
  1. causal dependencies vs. previously committed transactions at the time a transaction began,
  2. previous reads executed by the same transaction.
- Vector clocks used to establish visible snapshots.
High Level Overview (i)

- Transactions commit using a vector clock.
- Each node stores a log of committed vector clocks.

**Initial view of the visible snapshot**
- Upon a transaction T begins on N: it acquires the most recent vector clock in N’s commit log.

**View extension of the visible snapshot**
- Upon T reads on a node N:
  - T’s vector clock can be modified according to N’s commit log.
  - Three reading rules are applied using T’s vector clock.
Write operation
- Upon a transaction T writes V on data item O: it inserts <O,V> in T’s write-set.

Commit operation
- Read-only transactions always commit.
- Update transactions run a genuine 2-Phase Commit:
  - Upon prepare message reception (participant-side)
    - acquire read/write locks and validate read-set,
    - send back a tentative commit vector clock.
  - If all replies are positive (coordinator-side)
    - multicast write-set and final commit vector clock.
Rule 1: Reading Lower Bound

Node 0

Node 1 (it stores X)

Node 2 (it stores Y)

Most recent VC in VCLog

T_1: R(X)

T_1: VC

T_0: W(X,v)

T_0: W(Y,w)

T_0: Commit

X_{(2)}

Y_{(2)}

Commit

(1,1,1)

(1,1,1)

(1,1,1)

(1,2,2)

(1,2,2)

(1,2,2)

(1,2,2)

(1,2,2)

(1,2,2)

(1,2,2)

Euro-TM Workshop on Transactional Memory (WTM 2012), Bern, Switzerland
Rule 2: Reading Upper Bound

Node 0

- $(1,1,1)$

- $T_1:R(X)$

- $T_1:VC$ $(1,1,1)$

- Most recent VC in VCLog

- $X_{(1)}$

Node 1 (it stores $X$)

- $(1,1,1)$

- $X_{(1)}$

- $X_{(3)}$

- $T_0:W(X,v)$

- $T_0:W(Y,w)$

- $T_0:Commit$

- Commit

Node 2 (it stores $Y$)

- $(1,1,1)$

- $Y_{(1)}$

- $Y_{(2)}$

- $Y_{(3)}$

- Commit

- $T_1:R(Y)$

- $T_1:VC$ $(1,1,2)$

- $T_1:Commit$
Rule 3: Selection of Data Versions

- Informally: observe the most recent consistent version of data item \( id \) on node \( i \) based on \( T \)'s history (previous reads).

- Formally: iterate over the versions of \( id \) and return the most recent one s.t.

\[
id\text{.version.VN} \leq T\text{.VC}[i]
\]
Building the commit Vector Clock

- Based on a variant of the Skeen’s total order multicast algorithm [SKEEN85].

- Intuition:
  - Serialize all-and-only conflicting transactions, tracking
    - direct and transitive conflict dependencies,
    - causal relationship
Consistency Criterion

- GMU ensures Extended Update Serializability:
  - Update Serializability [ICDT86] ensures:
    - 1-Copy-Serializability (1CS) on the history restricted to committed update transactions;
    - 1CS on the history restricted to committed update transactions and any single read-only transaction.
    - But it can admit non-1CS histories containing at least 2 read-only transactions.

- Extended Update Serializability [Adya99]:
  - ensures US property also to executing transactions;
  - analogous to opacity in STMs.
Experiments on private cluster

8 core physical nodes

TPC-C
- 90% read-only xacts
- 10% update xacts

- 4 threads per node

- moderate contention (15% abort rate at 20 nodes)
Thanks for the attention
References


