A Framework for Transactional Consistency Models with Atomic Visibility

Daniel Solomon
30/05/18
Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence
Overview

▪ Introduction
  ▪ Notations and Definitions
  ▪ Transactional Consistency Models
  ▪ Models Relationship
  ▪ Optimizations
  ▪ Operational Model Equivalence
Introduction

▪ Our main focus is databases

▪ What is a database?
  ▪ Database is a organized collection of data

▪ There are many types of databases
  ▪ We will talk about replicated databases
Replicated database maintains shared data between several replicas

A client may perform *transaction* in any replica

Updates will propagate between all replicas

Why replicated database?
- Availability
- Low latency
- Offline purpose
Introduction – Cont.

- Ideally, we would like that the use of replicas will be transparent.
- Formally, *serializability*
  - The database behaves as if it executed transactions serially on a non-replicated copy of the data.
- Inefficient!
- Low latency and Availability properties may be affected.
Transactions

- Transaction is a sequence of *events*, each event is a *read* or *write* operation
- Transaction may be committed or aborted
- Atomic Visibility
- We will use:
  - $x, y$ as database objects
  - $u, v, w$ as local variables
  - $txn$ is a transaction
Anomalies

- In weaker consistency model than Serializability, non-serial behavior might appear, we will call them anomalies

- For example,
  - $txn_1 = \{x.\text{write}(\text{post}); y.\text{write}(\text{empty})\} ||$
  - $txn_2 = \{u = x.\text{read}(); y.\text{write}(\text{comment})\} ||$
  - $txn_3 = \{v = x.\text{read}(); w = y.\text{read}()\}$
  - Under specific assumptions, $u = \text{post}, v = \text{empty}, w = \text{comment}$
Anomalies – Cont.

▪ The consistency model defines which anomalies might appear

▪ Different types of anomalies affects directly the semantic of the software that interacting with the database

▪ Up until now, the current consistency models are coupled with the internal implementation of the database

▪ Lack of generalization or rules when deciding which model to use
Declarative Models

- To deal with this problem, we propose a framework that is used to specify six different consistency models for replicated databases.
- Specifications are *declarative* – do not refer to the db internals.
- Allow reasoning at higher abstraction level.
Atomic Visibility

- Usually *atomic visibility* is guaranteed, causing that for any transaction $T$:
  - All $T$ events are visible at once
  - None of $T$ events are visible
- Thanks to *atomic visibility*, transactions become our atomic unit so we may talk about relations on whole transactions
Overview

▪ Introduction

▪ **Notations and Definitions**
  ▪ Transactional Consistency Models
  ▪ Models Relationship
  ▪ Optimizations
  ▪ Operational Model Equivalence
Notations

- $Obj = \{x, y, \ldots \}$, all of them integers
- $Op = \{\text{read}(x, n), \text{write}(x, n)|x \in Obj, n \in \mathbb{Z}\}$
- $EventId$ – a set of infinite indexes
- $\text{historyevent} = (i, o), i \in EventId, o \in Op$
- $WEvent_x = \{(i, \text{write}(x, n)|i \in EventId, n \in \mathbb{Z}, x \in Obj\}$
- $REvent_x = \{(i, \text{read}(x, n)|i \in EventId, n \in \mathbb{Z} x \in Obj\}$
- $HEvent_x = WEvent_x \cup REvent_x$
Definition 1 – Transaction & History

- A transaction $T$ is a pair $(E, po)$, where $E \subseteq HEvent$ is a finite, non-empty set of events with distinct identifier. The program order $po$ is a total order over $E$.
- A history $H$ is a (finite or infinite) set of transactions with disjoint sets of event identifiers.
- All transactions in a history are assumed to be committed.
Definitions

▪ Prefix-finite:
  ▪ Relation is prefix-finite if every element has finitely many predecessors in the transitive closure of the relation \( \{a | (a, b) \in Trans(R) \} \) is finite

▪ \( VIS \):
  ▪ \( T_1 \xrightarrow{VIS} T_2 \) or \( (T_1, T_2) \in VIS \), if the transaction \( T_2 \) is aware of the updates made by transaction \( T_1 \)

▪ \( AR \):
  ▪ \( T_1 \xrightarrow{AR} T_2 \) or \( (T_1, T_2) \in AR \), means that the version of objects written by \( T_2 \) supersede those written by \( T_1 \)

▪ \( AR \) is a completion of \( VIS \) into a total order relation
Definition 2 – Abstract Execution

- An abstract execution is a triple $A = (H, VIS, AR)$ where:
  - $H$ is a history
  - Visibility: $VIS \subseteq H \times H$
  - Arbitration: $AR \subseteq H \times H$ is a prefix-finite, total order relation
  - $AR \supseteq VIS$ ($\Rightarrow VIS$ is a prefix-finite, acyclic relation)
Example

- Causality Violation anomaly

(a) Causality violation

\[ T_1 \quad \text{write}(x, \text{post}) \rightarrow \text{write}(y, \text{empty}) \quad \text{VIS} \quad \text{AR} \quad T_2 \quad \text{read}(x, \text{post}) \rightarrow \text{write}(y, \text{comment}) \quad \text{VIS} \quad \text{AR} \quad T_3 \quad \text{read}(x, \text{empty}) \rightarrow \text{read}(y, \text{comment}) \]
Consistency Model

- A consistency model specification is a set of consistency axioms $\phi$ constraining executions.

- The model allows those histories for which there exists an execution that satisfies the axioms:
  - $Hist_\phi = \{H | \exists VIS, AR. (H, VIS, AR) \models \phi\}$
  - This set (or its complement) defines the anomalies in the consistency model $\phi$
Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence
Transactional Consistency Models

- We now describe 6 different consistency models
- Each model will be described by its axioms
- We start from the weakest model and we will strengthen them from one to another
(I) Read Atomic

- $\phi = \{Int, Ext\}$
- The weakest model we will see today
More Notations

- For a total order $R \subseteq A \times A$ and a set $A$, we let $\max^R(A)$ be the element $u \in A$ such that $\forall v \in A. v = u \lor (v, u) \in R$

- $R^{-1}(u) = \{v | (v, u) \in R\}$

- _ will be used for an irrelevant value
Internal Consistency

- Within the transaction, the database provides sequential semantics:
  - A read from an object returns the same value as the last write or read in this very transaction

\[
\forall (E, po) \in H. \forall e \in E. \forall x, n. (e = (_, \text{read}(x, n)) \land (po^{-1}(e) \cap HEvent_x \neq \emptyset)) \\
\implies \max_{po}(po^{-1}(e) \cap HEvent_x) = (_, (_, x, n)) \tag{INT}
\]

- *Unrepeatable reads* is disallowed as well:
  - if a transaction reads an object twice without writing to it in-between, it will read the same value in both cases
External Consistency

- We let \( T \vdash Write \, x : n \) if \( T \) writes to \( x \) and the last value written is \( n \):
  \[
  \max_{po}(E \cap WEvent_x) = (_, write(x, n))
  \]

- We let \( T \vdash Read \, x : n \) if \( T \) makes an external read from \( x \), before writing to \( x \) and \( n \) is the first value returned:
  \[
  \min_{po}(E \cap REvent_x) = (_, read(x, n))
  \]

- The value returned by an external read in \( T \) is determined by the transactions \( VIS \)-preceding \( T \) that write to \( x \)
  - If none exists, \( T \) reads the initial value 0

\[
\forall T \in \mathcal{H}. \forall x, n. T \vdash Read \, x : n \implies \\
((VIS^{-1}(T) \cap \{ S \mid S \vdash Write \, x : \_ \}) = \emptyset \land n = 0) \lor \\
\max_{AR}(VIS^{-1}(T) \cap \{ S \mid S \vdash Write \, x : \_ \}) \vdash Write \, x : n)
\]

(EXT)
Example – Internal Consistency

\[ Write(x, 1) \overset{po}{\rightarrow} Read(x, 1) \]

\[ Read(x, 0) \overset{po}{\rightarrow} Read(x, 0) \]
Example – External Consistency

(a) Causality violation

$T_1$  
write$(x, post) \xrightarrow{po} write(y, empty)$  
\(\text{VIS}\)  
\(\text{AR}\)  
\(\text{AR}\)

$T_2$  
read$(x, post) \xrightarrow{po} write(y, comment)$  
\(\text{VIS}\)  
\(\text{AR}\)  
\(\text{AR}\)

$T_3$  
read$(x, empty) \xrightarrow{po} read(y, comment)$

(c) Lost update

$T_1$  
acct := acct + 50

read$(acct, 0) \xrightarrow{po} write(acct, 50)$  
\(\text{VIS}\)  
\(\text{AR}\)  
\(\text{AR}\)

$T_2$  
acct := acct + 25

read$(acct, 0) \xrightarrow{po} write(acct, 25)$  
\(\text{AR}\)  
\(\text{VIS}\)

$T_3$  
read$(acct, 25)$
External Consistency – Cont.

- $Ext$ implies two more properties:
  - **No Dirty reads:**
    - A committed transaction cannot read a value written by an aborted or an ongoing transaction
    - A transaction cannot read a value that was overwritten by the transaction that wrote it
  - **Atomic Visibility:**
    - Either all or none of the transaction writes can be visible to another transaction
Read Atomic – Use Case

- Symmetric relation
- *Fractured Reads* anomaly

(a) Causality violation

(b) Fractured reads
(II) Causal Consistency

- $\phi = \{\text{Int}, \text{Ext}, \text{TransVis}\}$
- \textbf{TransVis}:
  - Requiring VIS to be transitive
Read Atomic & Causal Consistency

- Both can be implemented without requiring any coordination among replicas:
  - A replica can decide to commit a transaction without consulting others
  - Advantage: availability
- Lost Update: An anomaly they both can’t prevent
(III) Parallel Snapshot Isolation

- $\phi = \{\text{Int}, \text{Ext}, \text{TransVis}, \text{NoConflict}\}$

- **NoConflict:**
  - Disallows different transactions writing to the same object to be concurrent (prohibits *Lost Update* anomaly)
  - If two transactions write concurrently to an object, there must be a $\text{VIS}$ relation between them

\[
\forall T, S \in H. (T \neq S \land T \vdash \text{Write } x : \_ \land S \vdash \text{Write } x : \_ \rightarrow (T \xrightarrow{\text{VIS}} S \lor S \xrightarrow{\text{VIS}} T) \quad (\text{NoConflict})
\]
RA & CC & PSI

- Two concurrent transactions may be observed in different orders
- Long Fork:
(IV) Prefix Consistency

- $\phi = \{\text{Int}, \text{Ext}, \text{TransVis}, \text{Prefix}\}$
- **Prefix:**
  - If $T$ observes $S$, then it also observes all $AR$-predecessors of $S$
  - $AR; VIS \subseteq VIS$

![Diagram](image)
(V) Snapshot Isolation

- $\phi = \{\text{Int}, \text{Ext}, \text{Trans}Vis, \text{NoConflict}, \text{Prefix}\}$
- Prevents *Long Fork & Lost Update* anomalies
- Adopted by some major DB systems such as MongoDB, PostgreSQL, Oracle, MSSQL and many others.
- *Write Skew* anomaly:

```plaintext
if (acct1 + acct2 > 100)
    acct1 := acct1 - 100
    read(acct1, 60) \xrightarrow{po} read(acct2, 60) \xrightarrow{po} write(acct1, -40) T_1

if (acct1 + acct2 > 100)
    acct2 := acct2 - 100
    read(acct1, 60) \xrightarrow{po} read(acct2, 60) \xrightarrow{po} write(acct2, -40) T_2
```

(e) Write skew. Initially acct1 = acct2 = 60.
(VI) Serializability

- \( \phi = \{ \text{Int}, \text{Ext}, \text{TotalVis} \} \)

- TotalVis:
  - VIS relation must be total

(e) Write skew. Initially acct1 = acct2 = 60.

\[
\begin{align*}
\text{if } (\text{acct1} + \text{acct2} > 100) \\
\text{acct1} & := \text{acct1} - 100
\end{align*}
\]

\[
\begin{align*}
\text{if } (\text{acct1} + \text{acct2} > 100) \\
\text{acct2} & := \text{acct2} - 100
\end{align*}
\]
Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence
## Models Relationship

<table>
<thead>
<tr>
<th>Φ</th>
<th>Consistency model</th>
<th>Axioms (Figure 2)</th>
<th>Fractured reads</th>
<th>Causality violation</th>
<th>Lost update</th>
<th>Long fork</th>
<th>Write skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA</td>
<td>Read Atomic [6]</td>
<td>Int, Ext</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CC</td>
<td>Causal consistency [19, 12]</td>
<td>Int, Ext, TransVis</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PSI</td>
<td>Parallel snapshot isolation [24, 21]</td>
<td>Int, Ext, TransVis, NoConflict</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PC</td>
<td>Prefix consistency [13]</td>
<td>Int, Ext, Prefix</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SI</td>
<td>Snapshot isolation [8]</td>
<td>Int, Ext, Prefix, NoConflict</td>
<td>✗</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>SER</td>
<td>Serialisability [20]</td>
<td>Int, Ext, TotalVis</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

![Relationship Diagram]

**Figure 1** Consistency model definitions, anomalies and relationships.
Framework Benefits

- Declarative specifications
- High level relations
- Strengthening consistency is easy
Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- **Optimizations**
- Operational Model Equivalence
Optimizations

- Can we optimize an *abstract execution*?
- Since we speak about transactions and not low-level events, two different transactions may cause the same external behaviour.
- *Observationally Refines*:
  - $T$ observationally refines $S$, if we can replace $T$ with $S$ in the execution without invalidating the consistency axioms.
Observationally Refines – Cont.

▪ Context:
  ▪ Abstract execution with a “hole”
  ▪ $\chi = (H \cup \{[\ ]\}, VIS, AR), VIS, AR \subseteq (H \cup \{[\ ]\}) \times (H \cup \{[\ ]\})$
  ▪ $\chi[T] = (H \cup \{[T]\}, VIS[[\ ] \rightarrow T], AR[[\ ] \rightarrow T])$

▪ Formal definition:
  ▪ $T_1$ observationally refines $T_2$ on the consistency model $\phi$ ($T_1 \sqsubseteq_\phi T_2$) if
    $\forall \chi. \chi[T_1] \models \phi \Rightarrow \chi[T_2] \models \phi$
Optimizations – Cont.

- **Theorem 4**: Let $T_1, T_2$ be such that $(\{T_1, T_2\}, \emptyset, \emptyset) \models \text{Int}$
  - **RA**: We have $T_1 \sqsubseteq_{RA} T_2$ if and only if for all $x, n$:
    $$\neg (T_1 \vdash \text{Read } x: n) \Rightarrow \neg (T_2 \vdash \text{Read } x: n) \land (T_1 \vdash \text{Write } x: n \iff T_2 \vdash \text{Write } x: n)$$
  - **CC/PC/SER**: We have $T_1 \sqsubseteq_{\phi} T_2$ if and only if for all $x, n, m, l$:
    $$\neg (T_1 \vdash \text{Read } x: n) \Rightarrow (\neg (T_2 \vdash \text{Read } x: n) \land (T_1 \vdash \text{Write } x: n \iff T_2 \vdash \text{Write } x: n)) \land (T_1 \vdash \text{Write } x: m \Rightarrow m = n) \Rightarrow (T_2 \vdash \text{Write } x: l \Rightarrow l = n))$$
  - **SI/PSI**: We have $T_1 \sqsubseteq_{\phi} T_2$ if and only if for all $x, n$:
    $$T_1 \sqsubseteq_{CC} T_2 \land \neg (T_1 \vdash \text{Write } x: n) \Rightarrow (\neg (T_2 \vdash \text{Write } x: n))$$
Optimizations – Cont.

- Notice that since we defined *external reads* by $T \vdash Read \, x : \ldots$ and $T \vdash Write \, x : \ldots$, two transactions that have the same *last writes* and the same *initial reads* are considered as equivalent since their *external behavior* is exactly the same.
Overview

- Introduction
- Notations and Definitions
- Transactional Consistency Models
- Models Relationship
- Optimizations
- Operational Model Equivalence
Operational Models Equivalence

▪ Without any practical implementation, our axiomatic specifications may not describe a real database behavior
▪ We now prove that our abstract models are equivalent to operational ones
▪ It will be done by showing algorithms that are very close to actual implementations
The System

- The database consists of a set of replicas, $RId = \{r_0, r_1, \ldots \}$
- We assume that the system is fully connected
- All client operations in the same transaction are being executed in a specific replica
- Any transaction eventually terminates
  - Then the replica decides to abort or commit it
  - On commit, a transaction log broadcast message with the updates will be sent by the replica
Transaction Log

- $t$: $\rho$
  - $\rho \in \{write(x, n) | x \in \text{Obj}, n \in \mathbb{Z}\}^* \triangleq \text{UpdateList}$
  - $t \in \mathbb{N}$ is the unique timestamp
- $LogSet \triangleq \bigcup_{\text{unique } t} \text{TransactionLog}_t$
Replica State

- \( RState \triangleq LogSet \times (UpdateList \cup \{idle\}) \)
- The replica state is a pair \((D, l)\)
  - \(D\) is the database copy of \(r\), represented by the set of logs of committed transactions
  - \(l\) is either the sequence of updates done so far by a single running transaction or \textit{idle}
System Configuration

- $\text{Config} \triangleq (\text{RId} \rightarrow \text{RState}) \times \text{LogSet}$
- The configuration of the whole system is $(R, M) \in \text{Config}$
  - $R(r)$ is the state of replica $r$
  - $M$ is the pool of messages which are in transit among the replicas
- $\rightarrow$ transition relation is defined by $\text{Config} \times \text{LEvent} \times \text{Config}$
- $\text{LEvent}$ consists triples $(i, r, o)$ $i \in \text{EventId}, r \in \text{RId}, o \in \text{COp}$
- $\text{COp}$ is the set of all low level operations:
  - $\text{COp} = \{\text{start}, \text{read}(x, n), \text{write}(x, n), \text{commit}(t), \text{abort}, \text{receive}(t: \rho) | x \in \text{Obj}, n \in \mathbb{Z}, t \in \mathbb{N}, \rho \in \text{UpdateList}\}$
System Configuration – Transitions

- We now describe how each low-level operations changes the system configuration

- **Start**
  - **Start** may be operated only if the transaction is in *idle* state
  - In order to signify that the replica is executing a transaction we change *idle* to { }

\[
\text{(Start) } \quad e = (\_, r, \text{start}) \\
(R[r \mapsto (D, \text{idle})], M) \xrightarrow{e} (R[r \mapsto (D, \epsilon)], M)
\]
System Configuration – Transitions

▪ **Write**
  ▪ The record $write(x, n)$ is appended to the current sequence of updates

\[
\begin{align*}
(\text{Write}) & \\
& \quad \quad e = (\_ , r, write(x, n)) \\
& \quad \quad (R[r \leftrightarrow (D, \rho)], M) \xrightarrow{e} (R[r \leftrightarrow (D, \rho \cdot write(x, n))], M)
\end{align*}
\]
System Configuration – Transitions

▪ **Read**
  ▪ The returned value is determined by a *lastval* function
  ▪ *lastval* function is based on the maintained database copy or replica \( r \) and the current *UpdateList*
    ▪ Search in *UpdateList* for *write*(\( x, \_ \)) in reverse order
    ▪ Search in \( D \) for *write*(\( x, \_ \)) by descending order of the timestamps
    ▪ If no such *write*, a value 0 is returned

\[
\begin{align*}
\text{(Read)} & \quad e = (\_, r, \text{read}(x, n)) \quad n = \text{lastval}(x, D \cup \{\infty : \rho\}) \\
& \quad (R[r \mapsto (D, \rho)], M) \xrightarrow{e} (R[r \mapsto (D, \rho)], M)
\end{align*}
\]
System Configuration – Transitions

- **Abort**
  - If a transaction aborts at replica \( r \), the current sequence of updates is in \( r \) is cleared

\[
\begin{align*}
(Abort) & \\
\text{e} & = (\_, r, \text{abort}) \\
(R[r \leftrightarrow (D, \rho)], M) & \xrightarrow{e} (R[r \leftrightarrow (D, idle)], M)
\end{align*}
\]
System Configuration – Transitions

- **Commit**
  - If a transaction commits, it gets assigned a *timestamp* $t$ and its transaction log is added to the message pool.
  - $t$ must be a distinct timestamp and must be greater than all timestamps that $r$ is aware of.
  - A single message is sent for each commit, which ensures *atomic visibility* property.

\[
e = (\_, r, \text{commit}(t))
\]

\[(\forall r', D'. R(r') = (D', \_)) \implies (t : \_) \notin D' \quad (\forall t'. (t' : \_) \in D \implies t > t')
\]

\[
(R[r \mapsto (D, \rho)], M) \xrightarrow{e} (R[r \mapsto (D \cup \{t : \rho\}, \text{idle})], M \cup \{t : \rho\})
\]
System Configuration – Transitions

▪ **Receive**
  - A replica $r$ may receive a transaction log from the message pool, only if it is in *idle* state
  - The received transaction log is added to the database copy

\[
\text{(Receive)} \quad \frac{e = (_{\_}, r, \text{receive}(t : \rho))}{(R[r \mapsto (D, \text{idle})], M \cup \{(t : \rho)\}) \xrightarrow{e} (R[r \mapsto (D \cup \{(t : \rho)\}, \text{idle}]), M \cup \{t : \rho\})}
\]
System Configuration – Transitions – Cont.

- We define the semantics of the operational model by considering all sequences of transitions generated by \( \rightarrow \) starting from an initial configuration
  - Log sets of all replicas are empty
  - The message pool is empty
Concrete Execution

- **Concrete execution:**
  - Let \((R_0, M_0) = (\forall r. (\emptyset, idle), \emptyset)\). A concrete execution is a pair \(C = (E, <)\)
  - \(E \subseteq LEvent\), \(<\) is a prefix-finite, total order over \(E\)
  - let \((e_1, e_2, ...\) events in \(E\) ordered by \(<\), then for some configurations \((R_1, M_1), (R_2, M_2), ... \in Config\), we have
    - \((R_0, M_0) \xrightarrow{e_1} (R_1, M_1) \xrightarrow{e_2} (R_2, M_2) \xrightarrow{e_3} ...\)
Equivalence – Read Atomic

- We want to show that the operational model defined by the transition function indeed defines the semantics of Read Atomic model

- $TSC$: 
  - Function that maps read/write event to its committed transaction

\[
TSC(e) = \begin{cases} 
  t, & \text{if } \exists r. e \in \{(\_, r, \text{read}(\_, \_)), (\_, r, \text{write}(\_, \_))\} \land \\
  \exists g \in E. g = (\_, r, \text{commit}(t)) \land \\
  \neg(\exists f \in \{(\_, r, \text{commit}(\_)), (\_, r, \text{abort})\}. (e < f < g)) & \\
  \text{undefined, otherwise}
\end{cases}
\]
History

- We first map *concrete execution* into a history
- The history of $C = (E, \prec)$ is defined as follows:
  - $\text{history}(C) = \{T_t | \{e \in E | TS_C(e) = t\} \neq \emptyset\}$ where $T_t = (E_t, po_t)$
  - $E_t = \{(i, o) | \exists e \in E. e = (i, _, o) \land TS_C(e) = t\}$
  - $po_t = \{(i_1, o_1), (i_2, o_2) | (i_1, o_1), (i_2, o_2) \in E_t \land (i_1, _, o_1) \prec (i_2, _, o_2)\}$
Equivalence – Read Atomic – Cont.

- \( \text{history}(\text{ConcExec}_{RA}) = \text{Hist}_{RA} \)

- \( \text{ConcExec}_{RA} \) is the set of concrete executions satisfying the Read Atomic model constraints
Equivalence – Read Atomic – Proof Outline

▪ \( \text{history} (\text{ConcExec}_{RA}) \subseteq \text{Hist}_{RA} \)

▪ Let \( C = (E, <) \in \text{ConcExec}_{RA} \), our goal is to show that \( \text{history}(C) \in \text{Hist}_{RA} \)

▪ We build an abstract execution from \( C \):
  ▪ \( A = (\text{history}(C), VIS, AR) \)
  ▪ \( AR = \{(T_{t_1}, T_{t_2}) | t_1 < t_2\} \)
  ▪ \( VIS = \left\{ (T_{t_1}, T_{t_2}) | e_1 \in \{ (\_ , r, \text{commit}(t_1)) , (\_ , r, \text{receive}(t_1: \_)) \} \land e_2 = (\_ , r, \text{commit}(t_2)) \land e_1 < e_2 \right\} \)
This construction provides:

- $AR$ – lifts the order of timestamps to transactions
- $VIS$ – reflects message delivery

We can show that any *abstract execution* constructed from a *concrete execution* as above, satisfies $Int, Ext$ and hence $∈ Hist_{RA}$.
Example – Read Atomic

(b) Fractured reads

\[
\begin{align*}
T_1 & \xrightarrow{\text{po}} T_2 \\
\text{write}(x_{\text{Alice}}, \text{Bob}) & \xrightarrow{\text{po}} \text{write}(x_{\text{Bob}}, \text{Alice}) \\
\text{read}(x_{\text{Alice}}, \text{Bob}) & \xrightarrow{\text{po}} \text{read}(x_{\text{Bob}}, \text{empty}) \\
\end{align*}
\]

\[
\begin{align*}
\text{e} & = (\_, r, \text{start}) \\
(R[r \mapsto (D, \text{idle})], M) & \xrightarrow{\text{e}} (R[r \mapsto (D, \text{idle})], M) \\
\text{e} & = (\_, r, \text{write}(x, n)) \\
(R[r \mapsto (D, \rho)], M) & \xrightarrow{\text{e}} (R[r \mapsto (D, \rho \cdot \text{write}(x, n))], M) \\
\text{e} & = (\_, r, \text{read}(x, n)) \\
\begin{align*}
\text{n} & = \text{lastval}(x, D \cup \{\infty : \rho\}) \\
(R[r \mapsto (D, \rho)], M) & \xrightarrow{\text{e}} (R[r \mapsto (D, \rho)], M) \\
\end{align*} \\
\text{e} & = (\_, r, \text{abort}) \\
(R[r \mapsto (D, \rho)], M) & \xrightarrow{\text{e}} (R[r \mapsto (D, \text{idle})], M) \\
\text{e} & = (\_, r, \text{commit}(t)) \\
(R[r \mapsto (D, \rho)], M) & \xrightarrow{\text{e}} (R[r \mapsto (D \cup \{t : \rho\}, \text{idle})], M \cup \{t : \rho\}) \\
\text{e} & = (\_, r, \text{receive}(t : \rho)) \\
(R[r \mapsto (D, \text{idle})], M \cup \{t : \rho\}) & \xrightarrow{\text{e}} (R[r \mapsto (D \cup \{t : \rho\}, \text{idle})], M \cup \{t : \rho\})
\end{align*}
\]
Stronger Operational Models – Causal Consistency

- For the stronger models, we will explain how to fulfill the axioms by constraining the communication protocol between the replicas.

- **CausalDeliv:**
  - Implies TransVis, ensures that the message delivery is causal.
  - If a replica $r$ sends the transaction log of $t_2$ after it sends or receives the transaction log of $t_1$, then every other replica $r'$ will receive the log $t_2$ only after it receives or sends the log $t_1$.

\[
\begin{align*}
(e_1 \in \{ (\_, r, \text{receive}(t_1 : \_)), (\_, r, \text{commit}(t_1)) \} \land & e_2 = (\_, r, \text{commit}(t_2)) \land e_1 \prec e_2 \land r \neq r' \land \\
& f_2 = (\_, r', \text{receive}(t_2 : \_)) \implies (\exists f_1 \in \{ (\_, r', \text{receive}(t_1 : \_)), (\_, r', \text{commit}(t_1)) \}. f_1 \prec f_2)
\end{align*}
\]

(CausalDeliv)
Example – Causal Consistency

\[
(e_1 \in \{\langle _, r, \text{receive}(t_1 : _) \rangle, \langle _, r, \text{commit}(t_1) \rangle \} \land e_2 = \langle _, r, \text{commit}(t_2) \rangle \land e_1 \prec e_2 \land r \neq r' \land f_2 = \langle _, r', \text{receive}(t_2 : _) \rangle) \implies (\exists f_1 \in \{\langle _, r', \text{receive}(t_1 : _) \rangle, \langle _, r', \text{commit}(t_1) \rangle \}. f_1 \prec f_2
\]

(CausalDeliv)
Stronger Operational Models – Prefix Consistency

- **MonTS:**
  - Timestamps must agree with the order in which transactions commit
  \[
  (e_1 = (_, _, \text{commit}(t_1)) \land e_2 = (_, _, \text{commit}(t_2)) \land e_1 < e_2) \implies t_1 < t_2 \]  
  (MonTS)

- **TotalDeliv**
  - Each transaction access a database snapshot that is closed under adding transactions with timestamps smaller than the ones already present in the snapshot
  \[
  (g = (_, r, \text{start}) \land e_2 \in \{}(_, r, \text{commit}(t_2)), (_, r, \text{receive}(t_2 : _))\} \land f = (_, _, \text{commit}(t_1)) \land t_1 < t_2 \land e_2 < g) \implies (\exists e_1 \in \{}(_, r, \text{commit}(t_1)), (_, r, \text{receive}(t_1 : _))\}. e_1 < g
  \]  
  (TotalDeliv)

- Both can be implemented via a central server
- Together guarantee Prefix
Example – Prefix Consistency

\[
\begin{align*}
T_1 \text{ write}(x, post1) \leftrightsquigarrow & \text{ read}(x, post1) \xrightarrow{po} \text{ read}(y, empty) \\
T_2 \text{ write}(y, post2) \leftrightsquigarrow & \text{ read}(x, empty) \xrightarrow{po} \text{ read}(y, post2) \\
T_3 \text{ read}(x, post1) \xrightarrow{po} & \text{ read}(y, empty) \\
T_4 \text{ read}(x, empty) \xrightarrow{po} & \text{ read}(y, post2)
\end{align*}
\]

\[
(e_1 = (\_\_, \_\_, \text{commit}(t_1)) \wedge e_2 = (\_\_, \_\_, \text{commit}(t_2)) \wedge e_1 \prec e_2) \implies t_1 < t_2 \quad \text{(MonTS)}
\]

\[
(g = (\_\_, r, \text{start}) \wedge e_2 \in \{(\_\_, r, \text{commit}(t_2)), (\_\_, r, \text{receive}(t_2 : \_\_))\} \wedge f = (\_\_, \_\_, \text{commit}(t_1)) \wedge t_1 < t_2 \wedge e_2 < g) \implies (\exists e_1 \in \{(\_\_, r, \text{commit}(t_1)), (\_\_, r, \text{receive}(t_1 : \_\_))\}. e_1 < g)
\quad \text{(TotalDeliv)}
\]
Stronger Operational Models – Parallel Snapshot Isolation

- **ConflictCheck:**
  - Allows transaction $T_1$ to commit at replica $r$ only if it passes a conflict detection check:
  - if $T_1$ updates an object $x$ that is also updated by a transaction $T_2$ committed at replica $r'$, then the replica $r$ must have received the log of $T_2$
  - If the check fails, $r$ must abort the transaction

\[
\begin{align*}
(e_1 &= (_, r, \text{write}(x, _)) \land f_1 = (_, r, \text{commit}(t_1)) \land \text{TS}_C(e_1) = t_1 \land \\
(e_2 &= (_, r', \text{write}(x, _)) \land f_2 = (_, r', \text{commit}(t_2)) \land \text{TS}_C(e_2) = t_2 \land f_2 < f_1 \land r \neq r') \\
\implies (\exists g \in E. g = (_, r, \text{receive}(t_2 : _)) \land g < f_1),
\end{align*}
\]

- May be implemented by requiring replica to coordinate with others before a commit

68
Example – Parallel Snapshot Isolation

\[ T_1 \text{ acct := acct + 50} \]

\[ \text{read(acct, 0)} \xrightarrow{\text{po}} \text{write(acct, 50)} \]

\[ \text{read(acct, 0)} \xrightarrow{\text{po}} \text{write(acct, 25)} \]

\[ T_2 \text{ acct := acct + 25} \]

\[ \text{VIS} \]

\[ \text{VIS} \]

\[ T_3 \]

\[ \text{read(acct, 25)} \]

\[ (e_1 = (_, r, \text{write}(x, _)) \land f_1 = (_, r, \text{commit}(t_1)) \land TSC(e_1) = t_1 \land \]

\[ e_2 = (_, r', \text{write}(x, _)) \land f_2 = (_, r', \text{commit}(t_2)) \land TSC(e_2) = t_2 \land f_2 \prec f_1 \land r \neq r'\]

\[ \implies (\exists g \in E. g = (_, r, \text{receive}(t_2 : _)) \land g \prec f_1), \quad \text{(ConflictCheck)} \]
# Stronger Operational Models

<table>
<thead>
<tr>
<th></th>
<th>Constraints</th>
<th></th>
<th>Constraints</th>
<th></th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA</td>
<td>None</td>
<td>PSI</td>
<td>(CausalDeliv), (ConflictCheck)</td>
<td>SI</td>
<td>(MonTS), (TotalDeliv),</td>
</tr>
<tr>
<td>CC</td>
<td>(CausalDeliv)</td>
<td>PC</td>
<td>(MonTS), (TotalDeliv)</td>
<td></td>
<td>(ConflictCheck)</td>
</tr>
</tbody>
</table>
Conclusion

- We have proposed a framework for specifying transactional consistency models of replicated databases
- We derived 6 different models using the framework
- The models are declarative which gives us a better understanding (?) of the database behaviour and allows us to discuss about the relations between the transactions
- The declarative framework may be used to prove correctness and specify optimizations in a more elegant and simpler way
- Using this framework we may create some new consistency models
- For database architecture designer, the framework helps to determine which model to use for maximum efficiency
Thank You!