Formal Development of Fault Tolerant Transactions for a Replicated Database using Ordered Broadcasts

Divakar Yadav and Michael Butler

Dependable Systems and Software Engineering
School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ UK

Outline

- Background
- System Model
- Ordering properties
- Outline of Event B development of ordered broadcast system
- Conclusions
Background

- Distributed System is a collection of *autonomous computers* spatially separated
- Advantage of Data Replication when Transactional Workload is predominantly read only
- Complexity of keeping data identical during updates due to Conflicting Transactions
- A Distributed Transaction spans multiple sites reading or updating data.
- *Read Anywhere Write Everywhere* based replica control requires transaction to read one copy and write all copies

Some Issues …

- Atomic Commitment of Distributed Transaction
- Race Conditions among Conflicting Transactions
- Messaging Infrastructure
- One Copy Equivalence consistency criterion
Event B

- Event B provides complete framework for developing mathematical model of distributed algorithms by
  - Rigorous description of problem.
  - Gradually introducing solution in refinement steps.
  - Verification of correctness of solution by discharging proof obligations.

- Atelier B, Click’n’Prove, B Toolkit provides support for discharge of proof obligation through automatic and interactive prover.

- Goals of Refinement
  - To understand more and more of system in incremental way
  - To distribute complexity of problem
  - To prove that concrete behaviour are similar to abstract one
  - To distribute the complexities of proofs
System Model

- Transaction $T_i$ is defined over $objectset[T_i]$ and $writeset[T_i]$ where,
  - $objectset[T_i] = \text{set of data objects to read/write}$
  - $writeset[T_i] = \text{set of data objects to be updated}$
  - $writeset[T_i] \subseteq objectset[T_i]$ and $objectset[T_i] \neq \emptyset$

- Transaction Types
  - Read-Only Transactions ($writeset[T_i] = \emptyset$)
  - Update Transactions ($writeset[T_i] \neq \emptyset$)

- Conflicting Update Transactions
  Update transactions $T_i$ and $T_j$ defined over data object sets $objectset[T_i]$ and $objectset[T_j]$
  conflicts if $objectset[T_i] \cap objectset[T_j] \neq \emptyset$

- Commitment of Transactions (Replicated Database)
  - Read-Only Transactions always commits
  - Update globally commit if it commits at all Participating Sites.
  - Update globally aborts if it Aborts at any Participating Site.
  - Abort at a site may be because of race conditions, time out, temporary data unavailability, deadlocks etc.
Sites contains the replica of data object.
System Model
Replicated Database

- Initial value of database is U.
Transaction $T_i$ is submitted at site $S_i$. 
Transaction $T_i$ is issued at site $S_i$. 

**System Model**

*Replicated Database*
Site $S_i$ sends messages to participating sites.
System Model
Replicated Database

Site $S_i$ sends messages to participating sites.
System Model

CASE 1: When All Site Pre-Commits

Distributed Transaction $T_i$ commits only if all Sub transactions commits.
System Model

CASE 1: When All Site Pre-Commits

- Distributed Transaction $T_i$ commits only if all Sub transactions commits.
If Distributed Transaction $T_i$ Commits, it commits at all sites.

$\implies$ All replicas are updated.
System Model

CASE 2: When at least one transaction fails at some site

Distributed Transaction $T_i$ aborts if Any Sub transactions aborts.
CASE 2: When at least one transaction fails at some site

Distributed Transaction $T_i$ Aborts if *Any* Sub transactions aborts.
If Distributed Transaction $T_i$ Aborts, it aborts at all sites.

$\implies$ None of replica is updated.
Overview of Events

Coordinator

StartTran
IssueWriteTran
Global Commit
Global Abort

Update Request
Vote Commit/Abort
Commit/Abort Decision

Participating Sites

BeginSubTran
LocalAbortTx / LocalCommitTx
ExecCommitDecision / ExecAbortDecision
**Reliable Broadcast**

A Reliable Broadcast satisfies the following properties:

**Validity**
If a correct process broadcasts a message $m$, then it eventually delivers $m$.

**Agreement**
All correct processes deliver the same set of messages, i.e., if a process delivers a message $m$ then all correct processes eventually deliver $m$.

**Integrity**
No spurious messages are ever delivered, i.e., for any message $m$, every correct process delivers $m$ at most once, and only if $m$ was previously broadcast by sender($m$).

In a Reliable Broadcast, there are no assumptions on maximum message delay, clock drift, delivery order, or causality.
Blocking of Update Transactions

- In our model the sites communicate by a **Reliable Broadcast**.

- A Reliable Broadcast **eventually** deliver the messages and does not impose any restriction on the **order** in which messages are delivered to sites.

- The Update Transactions may be **blocked** in following scenario:

  - S<sub>i</sub> starts transaction T<sub>i</sub> and acquire locks on objectset[T<sub>i</sub>] at site S<sub>i</sub>. Site S<sub>i</sub> broadcast update messages of T<sub>i</sub> to participating sites.
  
  - Another site S<sub>j</sub> starts a transaction T<sub>j</sub>, acquire locks on objectset[T<sub>j</sub>] at site S<sub>j</sub> and broadcast update messages of T<sub>j</sub> to participating sites.
  
  - The site S<sub>i</sub> delivers update message of T<sub>j</sub> from S<sub>j</sub> and S<sub>j</sub> delivers update message of T<sub>i</sub> from S<sub>i</sub>.
  
  - The T<sub>j</sub> is blocked at S<sub>i</sub> as S<sub>i</sub> waits for vote-commit from S<sub>j</sub> for T<sub>i</sub>. Similarly, T<sub>i</sub> is blocked at S<sub>j</sub> waiting for vote-commit from S<sub>i</sub> for T<sub>j</sub>. 

Avoiding the Blocking of Update Transactions

- In a replicated database that uses a Reliable Broadcast without ordering guarantees, the conflicting operations of transactions originating from different sites may arrive at participating sites in different orders.

- Unordered delivery of messages may cause blocking of update transactions which results in unnecessary aborts by the timeouts.

- Unnecessary aborts may be avoided if all sites deliver the messages in the same order.
If a particular process broadcasts a message $M_1$ before it broadcasts a message $M_2$, then each recipient process delivers $M_1$ before $M_2$.

Message $M_1$ (---) shows violation of FIFO order.
If a process delivers a message $M_1$ before it broadcasts a message $M_2$, then each recipient process delivers $M_1$ before $M_2$.

- Message $M_1$ shows violation of global causal ordering.
Causal Order

Ordering Properties (3)

Delivery order: M1, M2, M3, M4

If the broadcast of a message \( M_1 \) causally precedes the broadcast of a message \( M_2 \), then no process delivers \( M_2 \) unless it has previously delivered \( M_1 \).

\[
M_1 \prec M_2 \quad M_2 \prec M_3 \quad M_1 \prec M_2 \land M_2 \prec M_3 \implies M_1 \prec M_3
\]
The agreement and total order requirements of total order broadcast requires that all processes deliver the same sequence of messages.
A reliable broadcast that satisfies both causal and a total order.

causality: M1 → M2  M2 → M3
Total Order but Not a Causal Order

Ordering Properties (6)

A reliable broadcast that satisfies a total order but not causal order

Violation of causality

\[ M2 \prec M3 : \text{Each process MUST deliver } M2 \text{ before } M3 \]
Relationship among the ordered broadcast

- A total order broadcast (also called atomic broadcast) is a reliable broadcast that satisfies total order requirements.

- A causal order broadcast is a reliable broadcast that satisfies causal order requirement.

- A causal order satisfies both FIFO and Local Order.

- A total order broadcast does not captures causality.
  \[\Rightarrow\] A TO broadcast does not guarantee causal order

- A causal order broadcast is free to deliver messages in any order if their broadcasts are not causally related.
  \[\Rightarrow\] A CO broadcast does not guarantee total order

- A reliable broadcast that satisfies both causal and total order is called total causal order broadcast (also called causal atomic broadcast).
Causal Order Broadcast (Abstract Model)

Broadcast \((pp \in \text{PROCESS}, \ mm \in \text{MESSAGE})\) \equiv

\[
\begin{align*}
\text{WHEN} & \quad mm \notin \text{dom}(sender) \\
\text{THEN} & \quad \begin{align*}
order & := order \cup ( (sender-\{pp\} \times \{mm\}) \cup ( \text{deliver}[\{pp\}] \times \{mm\}) ) \\
\mid & \mid \\
\text{sender} & := \text{sender} \cup \{mm \mapsto pp\}
\end{align*}
\text{END};
\end{align*}
\]

Deliver \((pp \in \text{PROCESS}, \ mm \in \text{MESSAGE})\) \equiv

\[
\begin{align*}
\text{WHEN} & \quad mm \in \text{dom}(sender) \\
& \quad \wedge (pp \mapsto mm) \notin \text{deliver} \\
& \quad \wedge \forall m. (m \in \text{MESSAGE} \wedge (m \mapsto mm) \in order \Rightarrow (pp \mapsto m) \in \text{deliver}) \\
\text{THEN} & \quad \begin{align*}
\text{deliver} & := \text{deliver} \cup \{pp \mapsto mm\} \\
\mid & \mid \text{delorder}(pp) := \text{delorder}(pp) \cup ( \text{deliver}[\{pp\}] \times \{mm\})
\end{align*}
\text{END}
\end{align*}
\]
Incremental development of a system of causal order broadcast in Event B

[L1] This consist of abstract model of reliable broadcast. In this model a process communicate by broadcast and messages are delivered to the processes only once.

[L2] This is a refinement of abstract model which introduces causal ordering on the messages. In this refinement we outline how an abstract causal order is constructed by the sender.

[L3] In this refinement we introduce the notion of vector clocks. The abstract causal order is replaced by the vector clocks rules. We also discover gluing invariants which defines the relationship of abstract causal order and vector rules.

[L4] In this refinement steps we present the simplification of vector rules updating the vector clock of recipient process. Here we show that instead of updating whole vector of recipient process, a single value is updated.

[L5] This is a simple refinement further simplifying of causal deliver event.
Total Order Broadcast (Abstract Model)

Broadcast \((pp \in \text{PROCESS}, \, mm \in \text{MESSAGE})\)

\[
\text{WHEN } \quad mm \in \text{dom}(\text{sender}) \\
\text{THEN } \quad \text{sender} := \text{sender} \cup \{mm \mapsto pp\} \\
\text{END};
\]

Order \((pp \in \text{PROCESS}, \, mm \in \text{MESSAGE})\)

\[
\text{WHEN } \quad mm \in \text{dom}(\text{sender}) \\
\quad \land \, mm \in \text{ran}(\text{deliver}) \\
\quad \land \, \text{ran}(\text{deliver}) \subseteq \text{deliver} \{pp\} \\
\text{THEN } \quad \text{deliver} := \text{deliver} \cup \{pp \mapsto mm\} \\
\quad \| \, \text{totalorder} := \text{totalorder} \cup (\text{ran}(\text{deliver}) \times \{mm\}) \\
\quad \| \, \text{delorder}(pp) := \text{delorder}(pp) \cup (\text{deliver}[\{pp\}] \times \{mm\}) \\
\text{END}
\]

TODeliver \((pp \in \text{PROCESS}, \, mm \in \text{MESSAGE})\)

\[
\text{WHEN } \quad mm \in \text{dom}(\text{sender}) \\
\quad \land \, mm \in \text{ran}(\text{deliver}) \\
\quad \land \, (pp \mapsto mm) \in \text{deliver} \\
\quad \land \, \forall m. (m \in \text{MESSAGE} \land (m \mapsto mm) \in \text{totalorder} \Rightarrow (pp \mapsto m) \in \text{deliver}) \\
\text{THEN } \quad \text{deliver} := \text{deliver} \cup \{pp \mapsto mm\} \\
\quad \| \, \text{delorder}(pp) := \text{delorder}(pp) \cup (\text{deliver}[\{pp\}] \times \{mm\}) \\
\text{END}
\]
Mechanism to implement Total Order

Computation Message (M)

Control Message (M)
seqno(M)

P_1
Sequencer

P_2

P_3
Incremental development of a system of total order broadcast in Event B

[L1] This consist of abstract model of total order broadcast. In this model we outline how a total order is constructed.

[L2] This is a refinement of abstract model which introduces sequencer. In this refinement we show that the total order is built by the sequencer.

[L3] Through this refinement we precisely illustrate how a total order is build using the messages delivered to the sequencer.

[L4] In this refinement we introduce the notion of computation messages. Global sequence numbers of the computation message are generated by the sequencer. The delivery of the messages is done based on the sequence numbers.

[L5] In this refinement we introduce notion of control messages. We also introduce the relationship of each computation message with the control messages.

[L6] A new event Receive Control is introduced. We illustrate that a process other than sequencer can deliver a computation message only if it has received control message for it.
Incremental development of a system of total causal order broadcast in Event B

[L1] This level consists of abstract model of total causal order broadcast.

[L2] In this refinement step we replace abstract causal order and abstract total order with the vector clock rules and sequence numbers respectively.

[L3] In this refinement we show that sequence numbers are redundant and they can be replaced by the vector clock rules. We also present the gluing invariants relating abstract causal order, total order, vector clocks and sequence numbers.
Some Invariant Properties

Causal Order:

\[(m1 \rightarrow m2) \in causalorder \land (p \rightarrow m2) \in deliver \Rightarrow (m1 \rightarrow m2) \in delorder(p)\]

Total Order:

\[(m1 \rightarrow m2) \in delorder(p) \Rightarrow (m1 \rightarrow m2) \in totalorder\]
## Proof Statistics

<table>
<thead>
<tr>
<th>Development</th>
<th>Total POs</th>
<th>Completely automatic</th>
<th>Required Interaction</th>
<th>% proved automatically</th>
</tr>
</thead>
<tbody>
<tr>
<td>Causal Order Broadcast</td>
<td>80</td>
<td>54</td>
<td>26</td>
<td>67%</td>
</tr>
<tr>
<td>Total Order Broadcast</td>
<td>106</td>
<td>79</td>
<td>27</td>
<td>74%</td>
</tr>
<tr>
<td>Total Causal Order Broadcast</td>
<td>163</td>
<td>96</td>
<td>67</td>
<td>58%</td>
</tr>
</tbody>
</table>
Processing Transactions

Transaction may be processed over

- Reliable Broadcast
  (No ordering $\Rightarrow$ Blocking of Transactions)

- Causal Order Broadcast
  1. Captures causality among the transactions, deliver the transactions in the order they are caused.
  2. Transactions may be blocked if they are not causally related.
     (Solution: Static ordering on sites)

- Total Order Broadcast
  1. Guarantees the same delivery order at all sites $\Rightarrow$ No Blocking
  2. Causality among the transactions not preserved

- Total Causal Order Broadcast
  1. Guarantees the same delivery order at all sites $\Rightarrow$ No Blocking
  2. Causality among all the transactions is preserved.
Conclusions

- We outlined formal approach to modelling and analysing distributed transaction mechanism for replicated database using Event B.

- Updates are committed within the framework of commit protocol therefore global atomicity is achieved despite transaction failures.

- Group communication primitive allows to deliver message to the participating site in same order, thus avoid blocking of conflicting update transactions.

- Adequacy of Event B for modelling and analysing protocols for distributed databases.