Rethinking Serializable Multiversion Concurrency Control

Jose M. Falerio & Daniel J. Abadi
Yale University

Presenter: Suyash Gupta, Purdue University

January 24, 2017
Can we design a concurrency control protocol for multi-version database systems that is scalable?
Let's Refresh !!!

- Serializability
- Multiversion
- Timestamps
- Anti-dependency
**Basics I: Single vs. Multi-version Systems**

Single-version system:
- $T_r$ – Read $X$
- $T_w$ – Write $X$
- $X_0$

Multi-version system:
- $T_r$ – Read $X$
- $T_w$ – Write $X$
- $X_0$
Single-version system

- $T_r - \text{Read } X$
- $T_w - \text{Write } X$

$X_0$

$T_r$ and $T_w$ cannot simultaneously execute

Multi-version system

- $T_r - \text{Read } X$
- $T_w - \text{Write } X$

$X_0$

$X_1$

$T_r$ and $T_w$ can both simultaneously make progress
Multi-versioning buys more concurrency! Right?
**Basics II: Multi-versioning Example**

\[ T_0: \]
if savings + checking \( \geq 100 \)
savings \(-= 100\)

Savings: 100

Checking: 50

\[ T_1: \]
if savings + checking \( \geq 75 \)
checking \(-= 75\)

Constraint:
savings + checking \( \geq 0 \)
**Basics II**

**Basics II: Multi-versioning Example**

---

**T₀ commits**

**T₀:**
- if savings + checking \(\geq 100\)
- savings \(\leftarrow 100\)

---

**T₁ commits**

**T₁:**
- if savings + checking \(\geq 75\)
- checking \(\leftarrow 75\)

---

<table>
<thead>
<tr>
<th>Savings: 100</th>
<th>Savings: 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checking: 50</td>
<td>Checking: -25</td>
</tr>
</tbody>
</table>

**Constraint:**

savings + checking \(\geq 0\)
Use of conservative isolation techniques, similar to single-version systems can avoid such inconsistencies.
$T_0$:

```
savings += 100
```

- **begin:** start of the transaction
- **end:** end of the transaction

Monotonic timestamp generator

**Savings: 25**

- **begin:** 0
- **end:** 10

**Savings: 75**

- **begin:** 10
- **end:** infinite
Basics III: Multi-versioning Example II

$T_0$: savings += 100

- **Begin timestamp?**
- **Monotonic timestamp generator**

**Savings:** 25
- **Begin:** 0, **End:** 10

**Savings:** 75
- **Begin:** 10, **End:** $\text{inf}$
$T_0$: savings += 100

begin: 12  end:

| Savings: 25 | Savings: 75 |
| begin: 0    | begin: 10   |
| end: 10     | end: inf    |

Monotonic timestamp generator

Determines snapshot visible to txn
Basics III: Multi-versioning Example II

$T_0$:

savings += 100

End timestamp?

Monotonic timestamp generator

Savings: 25

begin: 0 end: 10

Savings: 75

begin: 10 end: inf
Basics III: Multi-versioning Example II

$T_0$: savings += 100

Determines visibility of txn’s writes

Monotonic timestamp generator

<table>
<thead>
<tr>
<th>Savings: 25</th>
<th>Savings: 75</th>
<th>Savings: 175</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin: 0</td>
<td>begin: 10</td>
<td>begin: 18</td>
</tr>
<tr>
<td>end: 10</td>
<td>end: 18</td>
<td>end: inf</td>
</tr>
</tbody>
</table>
$T_0$:
savings += 100

Global counter

Scalability bottleneck!

Monotonic timestamp generator

Savings: 175
Version management overhead

<table>
<thead>
<tr>
<th>Begin</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>inf</td>
</tr>
</tbody>
</table>

Savings: 25
Savings: 75
Savings: 175
Figure 1: Non-serializable interleaving, and corresponding serialization graph of $T_r$ and $T_w$. $r[x_1]$ denotes to a read of version 1 of record $x$, correspondingly, $w[x_1]$ denotes a write to record $x$, which produces version 1. A record’s subscript corresponds to the version read or written by the transaction.
Bohm

- Separate concurrency control from transaction execution.
- Concurrency control determines transaction order and version visibility.
- Execution performs logic given concurrency control ordering.
Transactions handed over to a single thread that maintains a log.

Position of a transaction in the log is its timestamp.

$m$ concurrency threads, each owning a logical partition, analyze each transaction in the log, and create a space “placeholder” if the transaction writes to any record in its partition.

$n$ separate execution threads, each execute a batch of transactions, and fill the pre-allocated spaces (write operation).
BOHM Diagrammatically

Concurrency Control

Determine legal schedule

Execution

Perform logic
Prior Multi-version CC systems assign each transaction, two timestamps, $t_{\text{begin}}$ and $t_{\text{end}}$.

- $t_{\text{begin}}$ – determines which versions of pre-existing records are visible.
- $t_{\text{end}}$ – determines the time at which writes become visible to other transactions.

BOHM assigns each transaction a single timestamp, $t_s$.

Each transaction appears to execute atomically at time $t_s$. 
Several threads contribute to the processing of a single transaction’s write-set.

BOHM assigns the responsibility of each record to one concurrency control thread.

When concurrency control layer receives a transaction, every concurrency control thread examines its write-set to determine whether any records belong to their partitions.
Version Field’s

- Start timestamp set to the timestamp of the transaction that creates the version.
- End timestamp is set to infinity,
- Txn pointer is set to the transaction that creates the version.
- Data is left uninitialized.
- Prev pointer is set to the preceding version of the record.
Batching

- Once a transaction has been processed by all threads it can be handed off to the transaction execution layer.

- One way – use synchronization barriers.

- BOHM avoids global coordination.

- Amortizes the cost of coordination across large batches of transactions.

- Each concurrency control thread receives an ordered batch of transactions.
• Partition data across multiple threads
• For every write, create a new version
Evaluating Transaction Logic

- Computation of version data occurs in this phase.

- Read dependencies may cause a transaction to wait as data may not be available.

- Write dependencies allow parallel execution

- Read-Modify-Write may require similar wait as Read.

- Versions can be Garbage Collected, as transactions are ordered in batches.
- Begins executing a batch after concurrency control completes
- Perform txn logic, write out data
• Begins executing a batch after concurrency control completes

• Perform txn logic, write out data

Replace txn reference with actual data
**Future Thoughts**

- BOHM’s biggest disadvantage is its need to pre-determine the write-sets of the transaction, prior to its execution.

- Interesting thought can be to design an approach on similar lines for on-line or real-time systems, with obvious tradeoffs.

- Batching transactions entering at same instant.
Thanx to all