Coordination Avoidance in Distributed Databases

2015

PhD thesis by Peter David Bailis
Outline

- Coordination avoidance and Invariant Confluence principle
- Examples of Invariant Confluence principle application
- Read-atomic multi-partition transactions
- Conclusions
Transaction serializability

- Transactions - groups of multiple operations over multiple data items
- Traditional way to deal with concurrent transactions - serializability: a database providing serializability guarantees that the result of executing the transactions is equivalent to some serial execution of the transactions
  - Convenient for programmers - no need to reason about concurrency
  - Inconvenient for databases - serializability requires coordination
- Reasons to avoid coordination - allows for greater availability of a distributed database, lower access latency and greater scalability
Thesis statement

- Key question posed by Bailis - when is it necessary to use coordination to achieve conflict-free parallel execution, and when is it possible to forego coordination without compromising the safety of parallel transactions

- Thesis Statement: Many semantic requirements of database-backed applications can be efficiently enforced without coordination, thus improving scalability, latency, and availability.
Application side: data invariants

- Bailis proposal: instead of reasoning about data consistency on the level of read and write operations (transactions), **consider what coordination is actually required by a given application**
- Make applications define data invariants and explicitly specify what correctness means to these applications
- Example: “each employee record is linked to a department record”
- Consider if **application requires coordination**, by taking into account both application invariants and the nature of data transformation in transaction
Invariant confluence test

- Invariant confluence determines whether the result of executing operations on independent copies of data can be combined (or “merged”) into a single, coherent (i.e., convergent) copy of database state.

- Given a set of operations, a safety property that we wish to maintain over all copies of database state, and a merge function, invariant confluence tells us whether coordination-free execution is possible.
Application of Invariant Confluence principle
I-Confluence proof construction

- To show a set of transactions are not invariant confluent with respect to an invariant I - use proof by counterexample: present two I-T-reachable states with a common ancestor that, when merged, are not I-valid.

- To show a set of transactions are invariant confluent with respect to an invariant I use proof by contradiction: show that if a state S is not I-valid, merging two I-T-reachable states S1 and S2 with a common ancestor state to produce S implies either one or both of S1 or S2 must not be I-valid.
Invariant confluence test example

- Key question: can invariants be violated by merging independent operations?

**Invariant:** User IDs are positive
**Operation:** Save new user
**Merge:** Add both records to DB

```
{}  --{}-- {}  --{}-- {}
add {Stu,ID=1}  add {Ann,ID=1}  add {Stu,ID=1}  add {Ann,ID=1}
MERGE
{{Stu,ID=1}, {Ann,ID=1}}
Invariant holds!
```

**Invariant:** User IDs are unique
**Operation:** Save new user
**Merge:** Add both records to DB

```
{}  --{}-- {}  --{}-- {}
add {Stu,ID=1}  add {Ann,ID=1}  add {Stu,ID=1}  add {Ann,ID=1}
MERGE
{{Stu,ID=1}, {Ann,ID=1}}
Invariant broken!
```

© Peter Bailis, MesosCon 2015 Keynote
### I-Confluence applied to common SQL operations

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Operation</th>
<th>invariant confluent?</th>
<th>Proof #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Equality</td>
<td>Any</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>Attribute Inequality</td>
<td>Any</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>Choose specific value</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>Choose some value</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>AUTO_INCREMENT</td>
<td>Insert</td>
<td>No</td>
<td>5</td>
</tr>
<tr>
<td>Foreign Key</td>
<td>Insert</td>
<td>Yes</td>
<td>6</td>
</tr>
<tr>
<td>Foreign Key</td>
<td>Delete</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>Foreign Key</td>
<td>Cascading Delete</td>
<td>Yes</td>
<td>8</td>
</tr>
<tr>
<td>Secondary Indexing</td>
<td>Update</td>
<td>Yes</td>
<td>9</td>
</tr>
<tr>
<td>Materialized Views</td>
<td>Update</td>
<td>Yes</td>
<td>10</td>
</tr>
<tr>
<td>&gt;</td>
<td>Increment [Counter]</td>
<td>Yes</td>
<td>11</td>
</tr>
<tr>
<td>&lt;</td>
<td>Increment [Counter]</td>
<td>No</td>
<td>12</td>
</tr>
<tr>
<td>&gt;</td>
<td>Decrement [Counter]</td>
<td>No</td>
<td>13</td>
</tr>
<tr>
<td>&lt;</td>
<td>Decrement [Counter]</td>
<td>Yes</td>
<td>14</td>
</tr>
<tr>
<td>[NOT] CONTAINS</td>
<td>Any [Set, List, Map]</td>
<td>Yes</td>
<td>15, 16</td>
</tr>
<tr>
<td>SIZE=</td>
<td>Mutation [Set, List, Map]</td>
<td>No</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 6.1: Example SQL (top) and ADT invariant confluence along with references to formal proofs in Section 6.2.

---

**Application of I-C principle**
I-Confluence example 1

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Operation</th>
<th>invariant confluent?</th>
<th>Proof #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniqueness</td>
<td>Choose specific value</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>Choose some value</td>
<td>Yes</td>
<td>4</td>
</tr>
</tbody>
</table>

- Claim: **common uniqueness invariants aren't I-Confluent** (e.g., PRIMARY KEY and UNIQUE constraints).
- Example invariant: user IDs must be unique
- However, **reads and deletions are both invariant confluent** under uniqueness invariants: **reading and removing items cannot introduce duplicates**
- Case 2: the database chooses unique values on behalf of users. **If replicas assign unique IDs within their respective portion of the ID namespace, then merging locally valid states will also be globally valid**

Proof by counterexample:

Invariant broken! © Peter Baik, MesseCon 2015 Keynote

Application of I-C principle
**I-Confluence example 2**

- Claim 9: writing arbitrary values is not invariant confluent with respect to multi-item uniqueness constraints.
- Proof: by counterexample

---

**Invariant: only one ops on staff at a time**

**Operations: change staffing**

- staff = {“Laura”:T, “Harry”:F, “Gary”:F}
- staff.set({“Laura”:F, “Harry”:T})
- staff.set({“Laura”:F, “Gary”:T})
- staff = {“Laura”:F, “Harry”:T, “Gary”:T}

 **Invariant violated!**
I-Confluence example 3

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Operation</th>
<th>invariant confluent?</th>
<th>Proof #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Equality</td>
<td>Any</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>Attribute Inequality</td>
<td>Any</td>
<td>Yes</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Claim:** Attributed equality invariants are i-confluent for any operations
- **Example:** *every user must have a last name assigned*, marking the LNAME column with a NOT NULL constraint
- **Proof by contradiction:** assume two database states S1 and S2 are each I-T-reachable under per-record inequality invariant I but that I(S1 U S2) is false. Then there must be a $r \in S1 U S2$ that violates I (i.e., $r$ has the forbidden value) and such $r$ must appear in S1, S2, or both. But, that would imply that one of S1 or S2 is not I-valid under I, a contradiction.
I-Confluence example 4

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Operation</th>
<th>invariant confluent?</th>
<th>Proof #</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO_INCREMENT</td>
<td>Insert</td>
<td>No</td>
<td>5</td>
</tr>
</tbody>
</table>

Consider the following transactions:

\[ T_{1s} := w(x_a = 1); \text{ commit} \]
\[ T_{2s} := w(x_b = 3); \text{ commit} \]

and the sequentiality constraint on records:

\[ I_s(D) = \{\max(r \in D) - \min(r \in D) = |D| + 1 \} \cup \{0\} \]

Now, \( I_s \) holds over the empty database \( (I_s(\{\}) \rightarrow \text{true}) \), while inserting sequential new records into independent, empty replicas is also valid:

\[ T_{1s}(\{\}) = \{x_a = 1\}, \quad I_u(\{x_a = 1\}) \rightarrow \text{true} \]
\[ T_{2s}(\{\}) = \{x_b = 3\}, \quad I_u(\{x_b = 3\}) \rightarrow \text{true} \]

However, merging these states results in invalid state:

\[ I_s(\{x_a = 1, x_b = 3\}) \rightarrow \text{false} \]

Therefore, \( \{T_{1s}, T_{2s}\} \) is not invariant confluent under \( I_s \).

- Claim 11: Writing arbitrary values are not invariant confluent with respect to sequentiality constraints.
- Proof: by counterexample

Overview
### I-Confluence example 5

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Operation</th>
<th>invariant confluent?</th>
<th>Proof #</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥</td>
<td>Increment [Counter]</td>
<td>Yes</td>
<td>11</td>
</tr>
<tr>
<td>&lt;</td>
<td>Increment [Counter]</td>
<td>No</td>
<td>12</td>
</tr>
<tr>
<td>&gt;</td>
<td>Decrement [Counter]</td>
<td>No</td>
<td>13</td>
</tr>
<tr>
<td>&lt;</td>
<td>Decrement [Counter]</td>
<td>Yes</td>
<td>14</td>
</tr>
</tbody>
</table>

**Claim 17** Counter ADT increments are invariant confluent with respect to greater-than constraints.

**Claim 18** Counter ADT increments are not invariant confluent with respect to less-than constraints.

**Claim 19** Counter ADT decrements are not invariant confluent with respect to greater-than constraints.

**Claim 20** Counter ADT decrements are invariant confluent with respect to less-than constraints.
I-Confluence example 6: foreign keys

- **Claim:** *Insertions under foreign key constraints are invariant confluent*

- **Proof by contradiction:**
  - Invalid state: a record missing a corresponding record on the opposite side of the association
  - $S1$ and $S2$ - correct states before the merge (no invalid records)
  - $r$ - invalid record in merged state $S$
  - As $S1$ and $S2$ are both valid, $r$ must have a corresponding foreign key record (f) that “disappeared” during merge. Merge (in the current model) does not remove versions, so this is impossible.

- Arbitrary **deletion/modification** of records is **unsafe**: a user might be added to a department that was concurrently deleted

---

Application of I-C principle
Read-atomic multi-partition transactions
Foreign keys updates

**How to achieve atomic visibility**

<table>
<thead>
<tr>
<th>X = 0</th>
<th>Y = 0</th>
</tr>
</thead>
</table>

**Strawman: Locking**

<table>
<thead>
<tr>
<th>X = 0</th>
<th>Y = 0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>X = 1</th>
<th>Y = 1</th>
</tr>
</thead>
</table>

I.e. in Facebook graph: either we’re both friends, or neither of us is

RAMP transactions
Read-atomic multi-partition transactions

RAMP TRANSACTIONS
DECOUPLE
ATOMIC VISIBILITY from
MUTUAL EXCLUSION

BASIC IDEA
LET CLIENTS RACE, but
HAVE READERS “CLEAN UP”

© Peter Bailis, "Coordination and the Art of Scaling", CloudFoundry 2014
Another problem solved with RAMP transactions

<table>
<thead>
<tr>
<th>Partition by primary key (ID)</th>
<th>How should we look up by age?</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Database" /> ID: 123 AGE: 22 ![Person]</td>
<td><img src="image2" alt="Database" /> ID: 412 AGE: 72 ![Old Person]</td>
</tr>
<tr>
<td><img src="image3" alt="Database" /> ID: 532 AGE: 42 ![Older Person]</td>
<td></td>
</tr>
<tr>
<td><img src="image4" alt="Database" /> ID: 892 AGE: 13 ![Child]</td>
<td><img src="image5" alt="Database" /> ID: 2345 AGE: 1 ![Child]</td>
</tr>
</tbody>
</table>

© Peter Bailis, "Scalable Atomic Visibility with RAMP Transactions", SIGMOD 2014
Secondary indexing

Partition by primary key (ID)

| ID: 123 | AGE: 22 |
| ID: 412 | AGE: 72 |

How should we look up by age?

| ID: 532 | AGE: 42 |
| ID: 892 | AGE: 13 |
| ID: 2345 | AGE: 1 |

SECONDARY INDEXING

Partition by primary key (ID)

How should we look up by age?

Option 1: Local Secondary Indexing
Build indexes co-located with primary data
WRITE ONE SERVER, READ ALL poor scalability

© Peter Bailis, “Scalable Atomic Visibility with RAMP Transactions”, SIGMOD 2014
Secondary indexing

Partition by primary key (ID)

<table>
<thead>
<tr>
<th>ID: 123</th>
<th>AGE: 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: 412</td>
<td>AGE: 72</td>
</tr>
</tbody>
</table>

How should we look up by age?

Option I: Local Secondary Indexing
Build indexes co-located with primary data
WRITE ONE SERVER, READ ALL
poor scalability

Option II: Global Secondary Indexing
Partition indexes by secondary key
WRITE 2+ SERVERS, READ ONE
scalable lookups

Real-world services employ either local secondary indexing (e.g., Espresso [38], Cassandra, and Google Megastore’s local indexes [7]) or non-atomic (incorrect) global secondary indexing (e.g., Espresso and Megastore’s global indexes, Yahoo! PANTS’s proposed secondary indexes [13]). The former is non-scalable but correct, while the latter is scalable but incorrect.

© Peter Bailis, 'Scalable Atomic Visibility with RAMP Transactions', SIGMOD 2014
Conclusions
Conclusions

Traditional database systems suffer from coordination bottlenecks

By understanding application requirements, we can avoid coordination unless necessary

We can build systems that actually scale while providing correct behavior

Use of validations (DB constraints) in Rails web-apps:

<table>
<thead>
<tr>
<th>Name</th>
<th>Occurrences</th>
<th>I-Confluent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>validates_presence_of</td>
<td>1762</td>
<td>Depends</td>
</tr>
<tr>
<td>validates_uniqueness_of</td>
<td>440</td>
<td>No</td>
</tr>
<tr>
<td>validates_length_of</td>
<td>438</td>
<td>Yes</td>
</tr>
<tr>
<td>validates_inclusion_of</td>
<td>201</td>
<td>Yes</td>
</tr>
<tr>
<td>validates_numericality_of</td>
<td>133</td>
<td>Yes</td>
</tr>
<tr>
<td>validates_associated</td>
<td>39</td>
<td>Depends</td>
</tr>
<tr>
<td>validates_email</td>
<td>34</td>
<td>Yes</td>
</tr>
<tr>
<td>validates_attachment_content_type</td>
<td>29</td>
<td>Yes</td>
</tr>
<tr>
<td>validates_attachment_size</td>
<td>29</td>
<td>Yes</td>
</tr>
<tr>
<td>validates_confirmation_of</td>
<td>19</td>
<td>Yes</td>
</tr>
<tr>
<td>Other</td>
<td>321</td>
<td>Mixed</td>
</tr>
</tbody>
</table>

Table 6.4: Use of and invariant confluence of built-in validations.
Thank you!