Consistency Rationing: *Pay only when it matters*

Tim Kraska, Martin Hentschel, Gustavo Alonso, Donald Kossmann
You own a Jewelry Store

- Items are highly valuable
- Any damage is expensive
- Requires protection
  - alarm systems / guards
  - insurance
  - security plans
- Items are handled carefully
  - Precise book-keeping
  - Demand planning etc.

It is expensive!!!
You own a Kiosk

- Items are not as valuable
- Strong protection is not required
- It is easier
- Less expensive
- It scales better

Of course, more can go wrong!
You own a Kiosk

- Items are not as valuable
- Strong protection is not required
- It is easier
- Less expensive
- It scales better

Protection Cost vs. Damage Cost

Of course, more can go wrong!
Consistency Rationing - Idea

- Strong consistency is expensive
- ACID prevents scaling & availability (CAP theorem)
- But not everything is worth gold!

Transaction Cost vs. Inconsistency Cost
1. Use ABC-analysis to categorize the data
2. Apply different consistency strategies per category
Cloud Computing and Consistency

Traditional architecture

- expensive hardware (mainframes)
- one or few machines (RAC)
- Single data center

Strong consistency requires:
- few messages between machines
  over fast interlinks

Cloud architecture

- COTS hardware
- thousands of machines
- Often multiple data center

Strong consistency requires:
- many messages across data centers (and expensive service calls)
Outline

- Consistency Rationing
- Adaptive Guarantees
- Implementation & Experiments
- Conclusion and Future Work
Outline

- Consistency Rationing
- Adaptive Guarantees
- Implementation & Experiments
- Conclusion and Future Work
### Consistency Rationing (Pre-Classification)

<table>
<thead>
<tr>
<th>Category</th>
<th>Characteristics</th>
<th>Guarantees</th>
<th>Use Case</th>
</tr>
</thead>
</table>
| **A-Data** | Inconsistencies are expensive and/or cannot be resolved | Serializable (2PL)  
  - Pessimistic CC as conflicts are expected  
  - No staleness: always up-to-date data | • Bank data  
• Atomic bomb |
| **B-Data** | Violations might be tolerable | Adaptive guarantees  
  - Switches between A & C guarantees  
  - Depends on some policy | • Product inventory  
• Tickets |
| **C-Data** | No inconsistency cost and/or inconsistency cannot occur | Session consistency  
  - Practical  
  - Still eventual consistent  
  - Allows for aggressive caching | • Recommendations  
• Customer profiles  
• Products |

In analogy to the ABC-analysis from Inventory Rationing
Consistency Rationing - Transactions

- Consistency guarantees per category instead of transaction level
  - Transactions are still allowed to overwrite consistency requirement
- Different categories can mix in a single operation/transaction
- For joins, unions, etc, the lowest category wins

- User Address (C-Data)
- Account (A-Data)

User address + account
- C-guarantees
- up-to-date account balance
- Eventual stale address
Outline

- Consistency Rationing
- Adaptive Guarantees
  - Implementation & Experiments
  - Conclusion and Future Work
Adaptive Guarantees for B-Data

- B-data: Inconsistency has a cost, but it might be tolerable
- Often the bottleneck in the system
- Here, we can make big improvements
- Let B-data automatically switch between A and C guarantees
- Use policy to optimize:

Transaction Cost vs. Inconsistency Cost
# B-Data Consistency Classes

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Use Cases</th>
<th>Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td>Non-uniform conflict rates</td>
<td>Collaborative editing</td>
</tr>
<tr>
<td><strong>Value Constraint</strong></td>
<td>• Updates are commutative</td>
<td>• Web shop</td>
</tr>
<tr>
<td></td>
<td>• A value constraint/limit exists</td>
<td>• Ticket reservation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Time-Based</strong></td>
<td>Consistency does not matter much until a certain moment in time</td>
<td>Auction systems</td>
</tr>
<tr>
<td><strong>Value-Based</strong></td>
<td>Consistency requirements depend on the data value</td>
<td>Plane-ticket reservation (business vs. economy class)</td>
</tr>
</tbody>
</table>
# B-Data Consistency Classes

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Use Cases</th>
<th>Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td>Collaborative editing</td>
<td><strong>General policy</strong></td>
</tr>
<tr>
<td>Non-uniform conflict rates</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| **Value Constraint** | • Updates are commutative  
  • A value constraint/limit exists | • Web shop  
  • Ticket reservation | • Fixed threshold policy  
  • Demarcation policy  
  • **Dynamic policy** |
| **Time-Based** | Auction systems | **Time-based policy** |
| Consistency does not matter much until a certain moment in time | | |
| **Value-Based** | Plane-ticket reservation  
  (business vs. economy class) | **Value-based policy** |
| Consistency requirements depend on the data value | | |
General Policy - Idea

Apply strong consistency protocols only if the likelihood of a conflict is high

1. Gather temporal statistics at runtime
2. Derive the likelihood of a conflict by means of a simple stochastic model
3. Use strong consistency if the likelihood of a conflict is higher than a certain threshold

Consistency becomes a probabilistic guarantee
General Policy - Model

- \( n \) servers
- Servers cache data with cache interval \( CI \)
- Load equally distributed
- Two updates considered as a conflict
- Conflicts for A and B data can be detected and resolved after every \( CI \)
- Every server makes local decisions (no synchronization)
1. Likelihood of a conflict inside one CI

\[
P_c(X) = P(X > 1) - \sum_{k=2}^{\infty} P(X = k) \left( \frac{1}{n} \right)^{k-1}
\]

2. Assuming further a Poisson process (simplification)
General Policy – Temporal Statistics

On every server: Collect update rate for a window $\omega$ with sliding factor $\delta$ per item

- Window size $\omega$ is a smoothing factor
- Sliding factor and window size influence adaptability and space requirement
General Policy – Setting the Threshold

- Use strong consistency if the savings are bigger than the penalty cost

\[
C_A - C_C > E_O(X) \\
C_A - C_C > P_C * C_O \\
\frac{C_A - C_C}{C_O} > P_C
\]

- Cost of A(CID) transaction \(C_A\)
- Cost of C transaction \(C_C\)
- Cost of inconsistency \(C_O\)
Value Constraint – Dynamic Policy

Apply strong consistency protocols only if the likelihood of violating a value constraint becomes high

- Commutative updates
- Without loss of generality the limit is assumed to be 0
- Assumptions as for the general policy
- Statistics: Value change over time
Value Constraint – Dynamic Policy

\[ P_C = P(T - Y < 0) \]

Y = Stochastic variable
T = Threshold

- Probability of Inconsistency $P_c$
- Threshold $T$
- Point $P_1$
- Point $P_2$
Outline

- Consistency Rationing
- Adaptive Guarantees
- Implementation & Experiments
- Conclusion and Future Work
Implementation

- Architecture of “Building a DB on S3” [Sigmod08]
  - Extended protocols
  - Additional services
    - Locking Service
    - Reliable Queues
    - Counter Service
  - Every call is priced
- Approach is not restricted to this architecture
  - PNUTS
  - Distributed DB
  - Traditional DB
Experiments

Modified TPC-W

- No refilling of the stock
- Different demand distributions
- Stock is uniformly distributed between 10-100
- Order mix (up to 6 items per basket (80-20 rule)
- 10 app servers, 1000 products
- $C_o = 0.01$ but up to 12,000 products are sold in 300sec
- Rationed consistency

Results represent just one possible scenario!!!
Overall Cost (including the penalty cost) per TRX [$/1000]

- A data
- C data
- Dynamic

Cost per transaction [$/1000]

- Uniform distribution
- 80-20 distribution
Overall Cost per TRX [$/1000]
Response Time [ms]
Conclusion and Future Work

- Rationing the consistency can provide big performance and cost benefits
- With consistency rationing we introduced the notion of probabilistic consistency guarantees
- Self-optimizing system – just the penalty cost is required
- Future work
  - Applying it to other architectures
  - Other optimization parameters (e.g. energy)
  - Better and faster statistics
Backup
Implementation - Statistics

- Servers make independent decisions
- Logical updates if possible
- General policy
  - Statistics stored at the record
  - High window size required
  - Assuming a conflict rate of less than 1% → λ ≈ 0.22
  - Allows to store all information per record in 48bit per record with a window size of 1 hour and a 5 min sliding factor

- Dynamic policy
  - Collects statistics only for hot records; all others are handled with a standard threshold
  - 10,000 hot records with 100 slices require 4MB of space.
Numeric Data – Value Constraint

Idea: Apply strong consistency protocols only if the likelihood of violating a value constraint becomes high.

Without loss of generality the limit is assumed to be 0.

Strategies:

- **Fixed threshold policy**
  \[ v - \Delta \leq T \]

- **Demarcation policy**
  - Every server gets a share of the value
  \[ T = v - \left\lfloor \frac{v}{n} \right\rfloor \]
  - No synchronization between servers → violations still possible

- **Dynamic policy**
  - Use temporal statistics