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

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 centers

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

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)

\[ P_\lambda(X = k) = \frac{\lambda^k}{k!} e^{-\lambda} \]

\[ P_c(X) = \left(1 - e^{-\lambda}(1 + \lambda)\right) - \sum_{k=2}^{\infty} \frac{\lambda^k}{k!} e^{-\lambda} \left( \frac{1}{n} \right)^{k-1} \]
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 $\delta$ is a multiple of CI
- Calculate average update rate $\bar{x}$ over all slices inside a window
- Derive the global state from local information $\lambda = \frac{\bar{x}n}{\delta}$

- $CI = 1$
- $\delta = 2$
- $\bar{x} = 1$
- $n = 4$
- $\lambda = 2$
General Policy – Setting the Threshold

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

$$C_A - C_C > E_0(X)$$
$$C_A - C_C > P_C \times C_O$$
$$\frac{C_A - C_C}{C_O} > P_C$$

- Cost of A(CID) transaction $C_A$
- Cost of C transaction $C_C$
- Cost of inconsistency $C_O$
Value Constraint

- Use Cases:
  - Web shop
  - Ticket reservation
- Commutative updates
- Consistency is defined by a value constraint (e.g., stock >= 0)
- Without loss of generality, the limit is assumed to be 0

\[ \Delta = -1 \]

\[ \begin{align*}
\text{Value Change} & \quad t \\
-3 & \quad -1 \\
-2 & \quad -2 \\
-4 & \\
\end{align*} \]

\[ \begin{align*}
\text{# Updates} & \quad t \\
2 & \quad 1 \\
1 & \quad 1 \\
2 & \\
\end{align*} \]
Value Constraint - Policies

Fixed threshold policy
Use strong consistency protocol if value drops below a fixed threshold $T$

$$v - \Delta \leq T$$

$v =$ Value of $attr_{k}$ of record $r$
$\Delta =$ Change by the current TRX

Demarcation policy
• Divide value among servers
• Every server gets a share of the value

$$T = v - \left\lfloor \frac{v}{n} \right\rfloor$$

• Adjust shares after $CI$ expires
• Still, doesn’t guarantee strong consistency
Value Constraint – Dynamic Policy

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

- Likelihood of a conflict is

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

\( T = \) Threshold when to switch
\( Y = \) Stochastic variable corresponding to the sum of \( \Delta \) within CI.
Value Constraint – Dynamic Policy

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

\( Y = \) Stochastic variable
\( T = \) Threshold
Dynamic policy – Temporal statistics

- Sliding factor $\delta$ is factor of $CI$ (not multiple)
  - The dynamic policy requires the variance
  - Interest is on hot spots $\rightarrow$ Less time required to gather statistics
- Convoluting the slices to derive the empirical PDF
  \[
  f_{CI} = f \ast f \ast \ldots \ast f \quad \text{CI/} \alpha \text{ times} \\
  f_{CI^n} = f_{CI} \ast f_{CI} \ast \ldots \ast f_{CI} \quad \text{n times}
  \]
- Determine the threshold by means of the CDF
  \[
  F_{CI^n}(T) > P_C(Y) \Rightarrow F_{CI^n}(T) > \frac{C_A - C_C}{C_O}
  \]
- Optimization: Use normal distribution instead of convolution
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
Step 1: Clients commit update records to pending update queues
Step 2: Checkpointing propagates updates from SQS to S3

[SIGMOD08]
Levels of Consistency/Availability

- Naïve (Shared Disk)
  - No Guarantees
- Basic
  - EC
- Atomicity
  - EC
- Monotonicity
  - Different Monotonicity Guarantees
- Advanced Atomicity
  - Monotonicity
- Snapshot Isolation
- 2PL
  - Serializability

Storage Service
  - + Simple Queue Service
  - + Locking Service (only for checkpointing)
  - + Advanced Queue Service
  - + Locking Service
  - + Advanced Counter Service
  - - Advanced Counter Service

EC = Eventual Consistent
Levels of Consistency/Availability

- Naïve (Shared Disk) No Guarantees
- Basic EC
- Atomicity EC
- Monotonicity Different Monotonicity Guarantees
- Advanced Atomicity Monotonicity
- Snapshot Isolation
- 2PL Serializability
- Storage Service
  - + Simple Queue Service
  - + Locking Service (only for checkpointing)
  - + Advanced Queue Service
  - + Locking Service
  - + Advanced Counter Service
  - - Advanced Counter Service

EC = Eventual Consistent

Availability:
- high
- low

Consistency:
- high
- low
Implementation - Statistics

- Servers make independent decisions
- Logical updates if possible
- General policy
  - Statistics stored at the record
  - Large window size required
  - Assuming a conflict rate of less than 1% → \( \lambda \approx 0.22 \) updates per CI
  - Assuming a window size of 1 hour and 5 min sliding factor: Allows to store the statistics in 48bit per record (4 bit per slide, with value 15 as infinite)
- Dynamic policy
  - Collects statistics only for hot records; all others are handled with a standard threshold or the General Policy
  - 10,000 hot records with 100 slices require 4MB of space.
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
- \( C_o = 0.01 \text{\$} \) 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 [$/1000]
Overall Cost per TRX [$/1000]
Response Time [ms]

![Graph showing response time for different policies of consistency: A data, C data, Dynamic. The graph compares uniform distribution with 80-20 distribution.](image-url)
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
  - Language support
  - Applying it to other architectures
  - Better and faster statistics
  - Consistency and analytics???