Consistency Rationing in the Cloud: *Pay only when it matters*

By
Sandeepkrishnan

Some of the slides in this presentation have been taken from http://www.cse.iitb.ac.in./dbms/CS632/Rationing.ppt

Introduction:

- Important Features Cloud Storage Solutions are Consistency, Scalability and Low cost.

- Provides different levels of consistencies.

- Higher consistency implies higher transactional cost, Lower consistency implies higher operational cost.

Why consistency rationing is needed?

- Not all data requires same level of consistency.

- Eg: Web Shop, Credit card and account balance requires higher consistency, user preferences like users who bought ‘X’ also bought ‘y’ requires less consistency.

- Also there is a cost associated with each level of consistency.

- Price of consistency is measured by number of services required to enforce it.

- In the same way inconsistency is measured by resulting percentage of incorrect operations that lead to penalty costs like losing a valuable customer etc.

What are we going to discuss here?

- A new dynamic consistency strategy.

- Reducing consistency requirements when possible and raising them when it matters.

How is it done?

- Data is divided in to three categories A, B and C. Each category is provided a different consistency level.

- A category contains data for which consistency violation results in higher penalty costs.

- C category contains data for which temporary inconsistency is acceptable.

- B category contains data whose consistency requirements vary over time.

Outline

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

Outline

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

Consistency Rationing (Classification)

Category	Characteristics	Guarantees	Use Case
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	• Recom-mendations • Customer profiles • Products

In analogy to the ABC-analysis from Inventory Rationing

Consistency Rationing (Category A)

- Provides Serializability in traditional sense.
- Always stays consistent
- In cloud – expensive
  - transaction cost.
  - Performance issues.
- Complex protocols needed
- Require more interaction with additional services (e.g., lock services, queuing services)
- Lower performance(response time).
- Serializability is provided using 2PL protocol.

Consistency Rationing (Category C)

- Session consistency.
- It is the minimum consistency level.
- Session of different clients will not always see each other's updates.
- Only uses Eventual consistency.
- Conflict resolution depends on type of updates.
- Non commutative updates (overrides) – last update wins.
- Commutative updates (numerical operations e.g., add) – conflict is resolved by applying updates one after other.
- It is very cheap.
- Permits extensive caching.
- Should always place data in C- category.

Consistency Rationing (Category B)

- Adaptive consistency.
- Switches between Serializability and Session consistency.
- Required level of consistency depends on the situation.

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.
- Here, we can make big improvements.
- Let B-data automatically switch between A & C categories.
- Use policy to optimize: Adaptive policies.

Transaction Cost vs. Inconsistency Cost

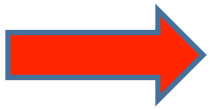
B-Data Consistency Classes

	Characteristics	Use Cases	Policies
General	Non-uniform conflict rates	Collaborative editing	General policy
Value Constraint	• Updates are commutative • A value constraint/ limit exists	• Web shop • Ticket reservation	• Fixed threshold policy • Demarcation policy • Dynamic policy
Time-Based	Consistency does not matter much until a certain moment in time	Auction systems	Time-based policy

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 an 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 implementing different levels of consistency.
- Servers cache data with cache interval - CI
- Within CI, C data is read.
- Two updates on same data are considered as a conflict.

The probability of conflicting update on a record is given by

$$P_c(X) = \underbrace{P(X>1)}_{(i)} - \underbrace{\sum_{k=2}^{\infty}\left(P(X=k)\left(\frac{1}{n}\right)^{k-1}\right)}_{(ii)}$$

✓ *X* is a stochastic variable corresponding to the number of updates to the same record within the cache interval.

✓ P(X > 1) is the probability of more than one update over the same record in one cache interval.

General Policy - Model

Conflicts can occur only if updates are issued on different servers, (ii) calculates the probability that the concurrent updates happen on the same server.

Assumption is made, that arrival of transactions is a Poisson process so above equation is modified around a single variable with mean arrival rate λ.

Probability density function of a Poisson distribution is given by

$$P_\lambda\left(X = k\right) = \frac{\lambda^k}{k!} e^{-\lambda}$$

General Policy - Model

$$P_c(X) = \underbrace{\left(1 - e^{-\lambda}(1+\lambda)\right)}_{(iii)} - \underbrace{\sum_{k=2}^{\infty}\left(\frac{\lambda^k}{k!}e^{-\lambda}\left(\frac{1}{n}\right)^{k-1}\right)}_{(iv)}$$

If n > 1, probability of conflict is less, second term of equation can be disregarded, hence following expression can be considered upper bound for conflict

$$P_C(X) = \left(1 - e^{-\lambda}(1+\lambda)\right)$$

General Policy – Setting the Threshold

- Use weak consistency if the savings of using weak consistency 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 transaction
- Cost of C transaction
- Cost of inconsistency (O)

Time Policies

- Based on a timestamp, increase consistency when timestamp is reached.
  Eg: online Auction systems.

- Simplest method is setting policies to a predefined value (eg. 5 minutes).

- Likelihood of conflict defined by Pc(Xt).

Numeric Data – Value Constraint

- Most of the conflicting updates cluster around numerical values.
- Eg: Stock of items in a store.
  Available tickets in a reservation system.
- Often characterized by integrity constraint defined as a limit (eg: stock > 0).

Fixed threshold policy

value is below a certain threshold, the record is handled under strong consistency

$$v - \Delta \leq T$$

Current value – amount <= Threshold

Issues :

- Inconsistency can occur if the sum of all updates on different servers let the value under watch drop below the limit.
- Finding the optimal Threshold - experimentally determine over time.
- Static threshold.

Demarcation policy

- Distribution among servers.
- Allowed to change local value (< local bound).
- Servers may request additional shares from other servers.
- Strong consistency required only if amount > local share is used.

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

Issues:

- Might not always ensure proper consistency.
- Skewed distribution of data Accesses.
- For large number of severs n, threshold tend towards v & B data is treated as A data.

Dynamic Policy

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

- Similar to general policy.

- The probability of the value of a record dropping below zero is calculated by using the formulae given in next slide .

- Here the P(T-Y < 0) refers to probability that the consistency constraint is violated, eg. Buying more items before the servers apply strong consistency.

Dynamic policy:
It implements probabilistic consistency guarantees by adjusting threshold.

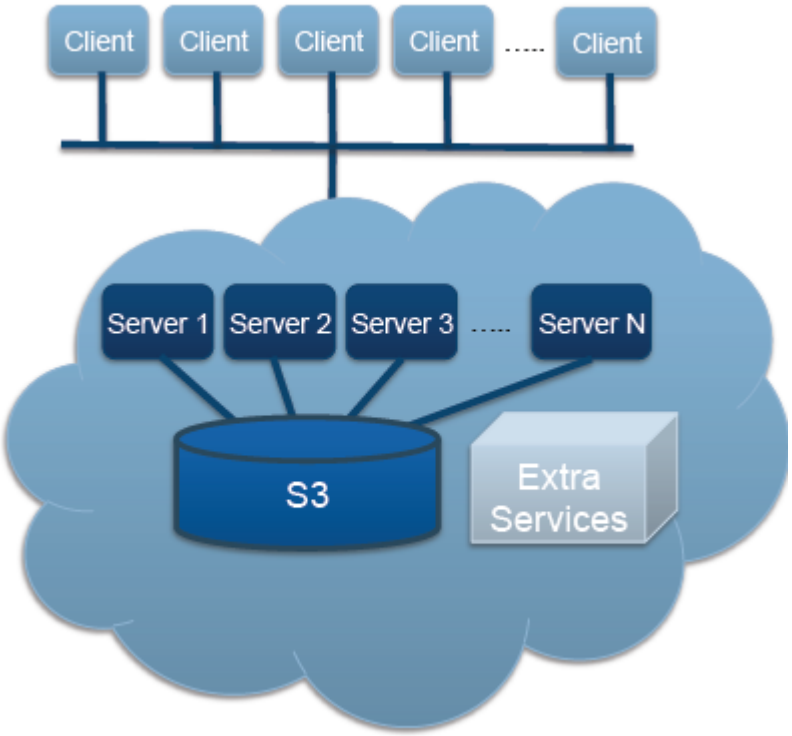
$$P_C(Y) = P(T - Y < 0)$$

Y is a stochastic variable corresponding to the sum of update values within the cache interval CI.

Outline

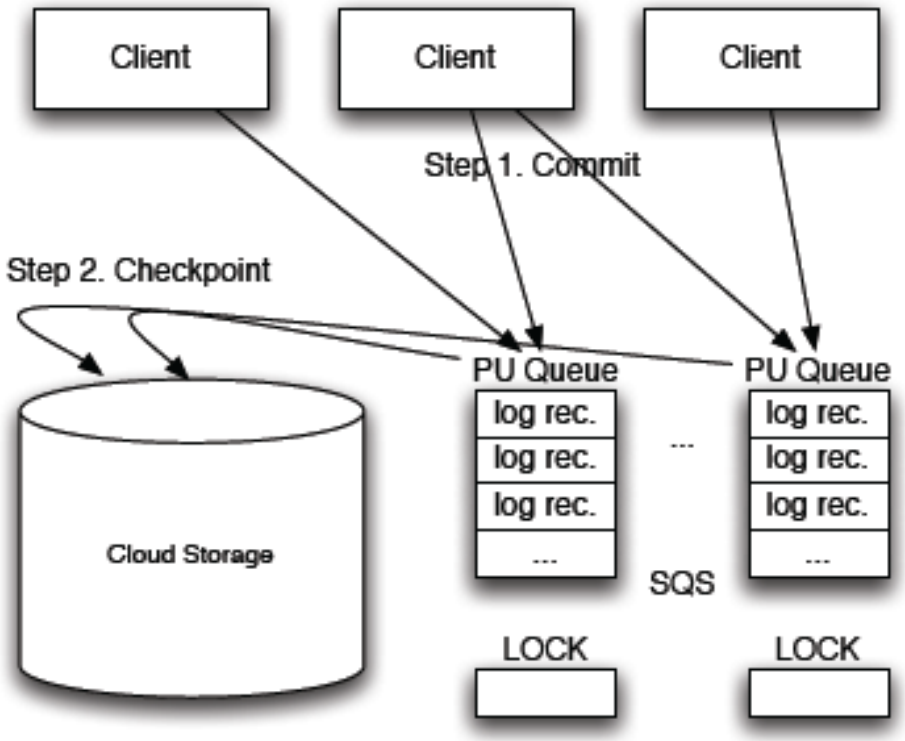
- Consistency Rationing
- Adaptive Guarantees
- **Implementation & Experiments**
- Conclusion and Future Work

Implementation - Architecture



- Architecture of “Building a DB on S3”.
- Extended protocols on top of S3.
- Additional services
  - Locking Service
  - Reliable Queues
  - Counter Service
- Logical logging.
- Metadata management, policies, Statistical component etc.

Implementation



Protocol Implementation

- Session consistency.
- Serializability.
- Meta data.
- Logical Logging.

Serializbility

- 2 Phase Locking protocol is used.
- Locking service is implemented to achieve exclusive access rights required by 2PL.
- Advanced Queue Service is implemented (better that SQS).
- Monotonicity protocol was simplified by increasing message id, storing only latest applied message id in a page.
- Pages with older message id indicates it need update.
- To make pages up to date all messages are retrieved, apply only messages with higher id than page id.

Session consistency

- Similar to that of S3.
- Protocol for read-your-write monotonicity. Keep highest timestamp for each page, if server detects old page from S3 can detect and re-read.
- Redirecting all requests from same client to same server inside a session is done using session id.
- Vector clocks are used to detect & resolve conflicts. Assigned to messages before sending them to queue's.

Logical Logging

- To allow higher rates of concurrency without inconsistencies.
- Logical log message contains ID of the modified record & operation performed to this record.
- To retrieve new state of a record, OP from PU Q is applied to item.
- Robust against concurrent updates for commutative operations. Updates are never lost because it is overwritten and all updates become visible independent of order.

Meta data

- Every collection contain metadata about its type.
- Consistency that need to be applied is determined based on which collection a record belongs to.
- For B data metadata contains name of the policy and parameters for it.

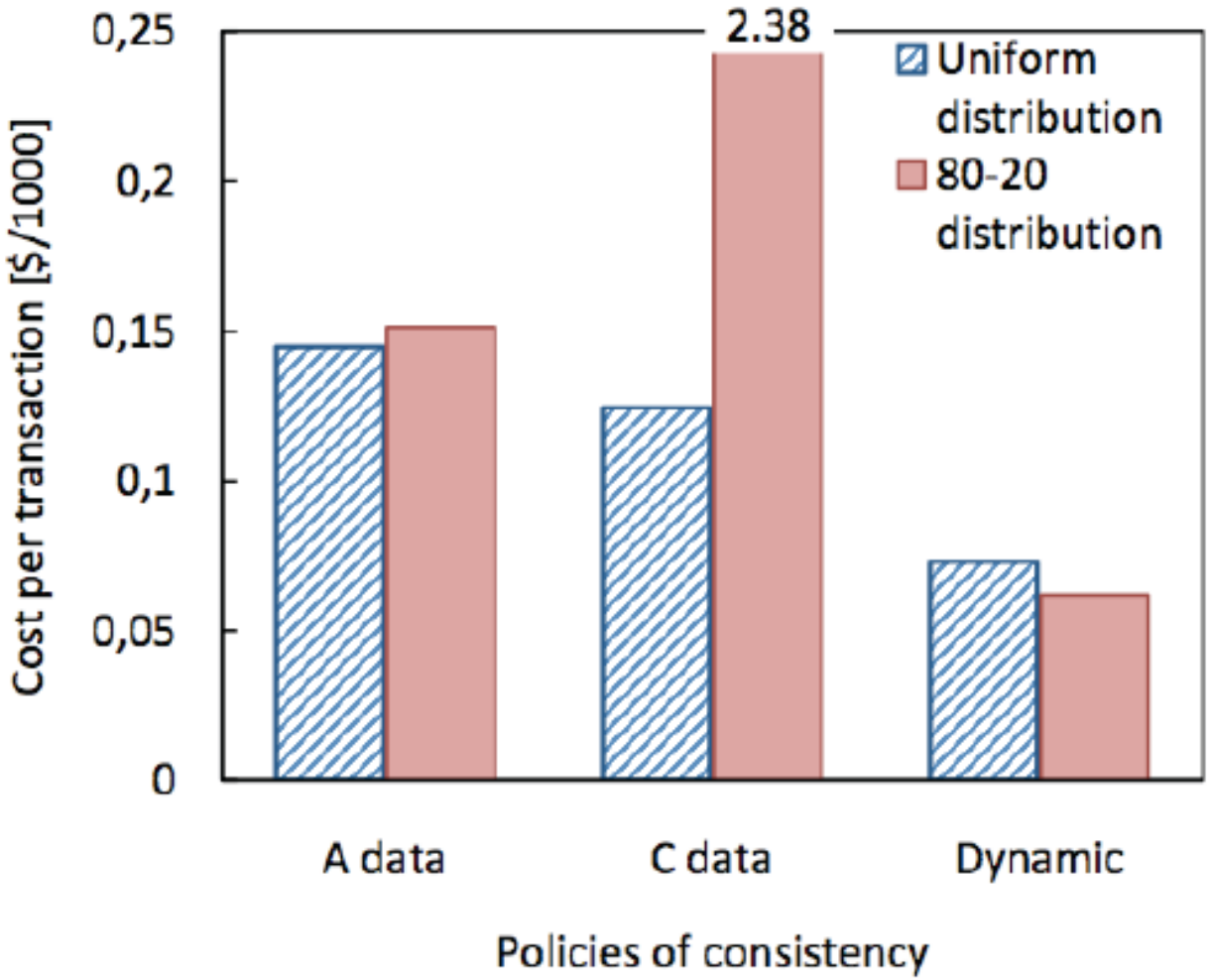
Experiments

Modified TPC-W

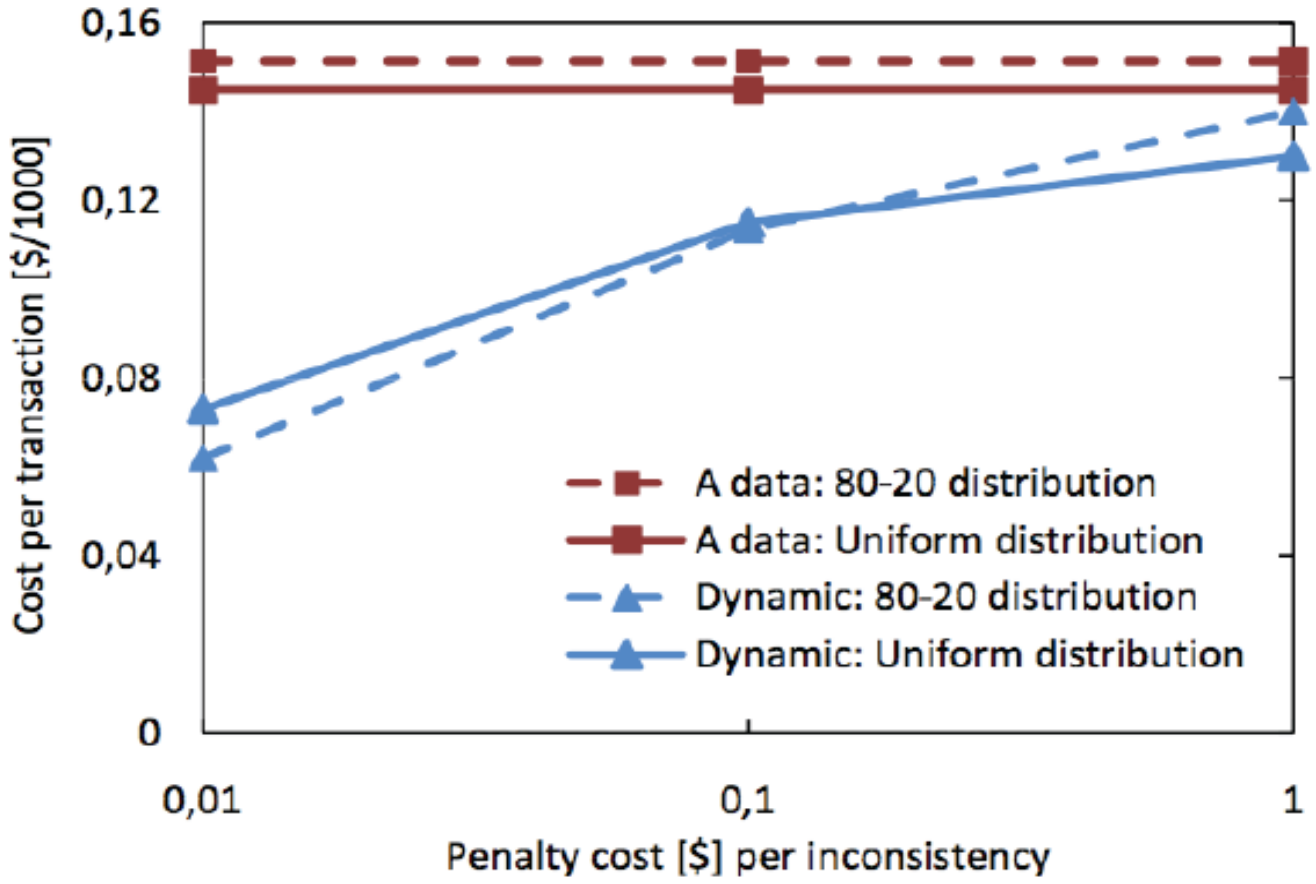
- Web shop – 14 different actions like product search, register a customer, purchase etc.
- Most write intensive mix is ordering mix in which 10% of actions are purchases, used most in experiments.
- Stock of each product between 10 and 100.
- One purchase action may request to buy several different products.
- 10 app servers, 1000 products
- Data categorization: first all as A data, then as C data and then a mix of data categories.

Results represent just one possible scenario!!!

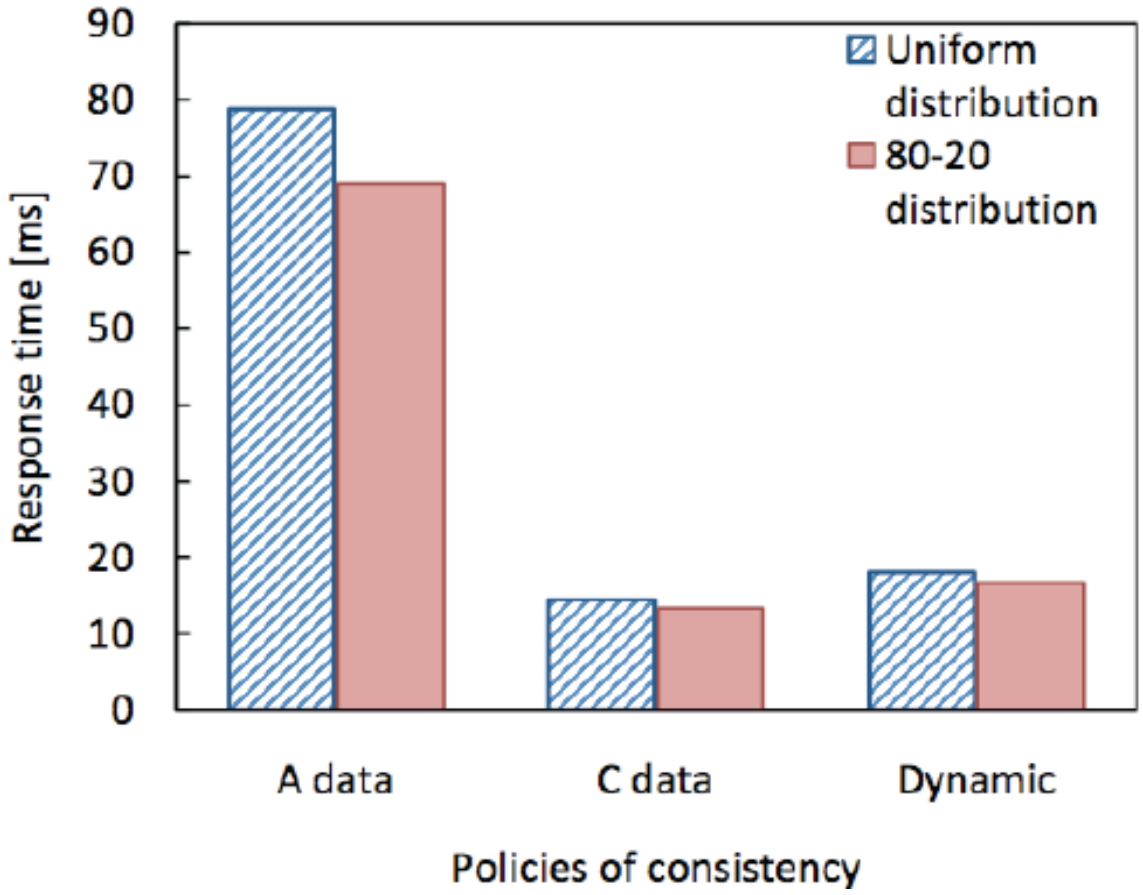
Overall Cost (including the penalty cost) per TRX [$/1000]



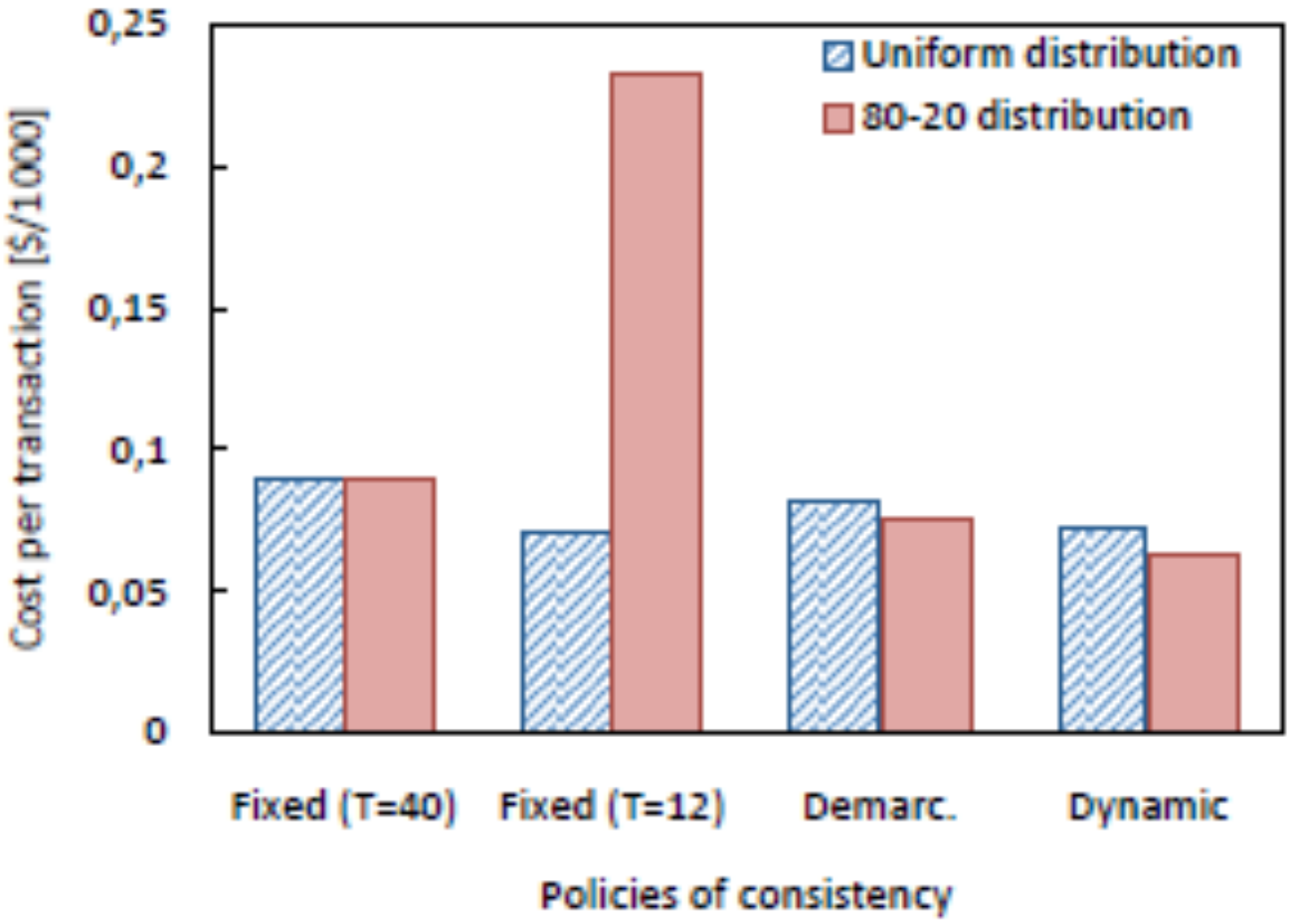
Overall Cost per TRX [$/1000], vary penalty costs



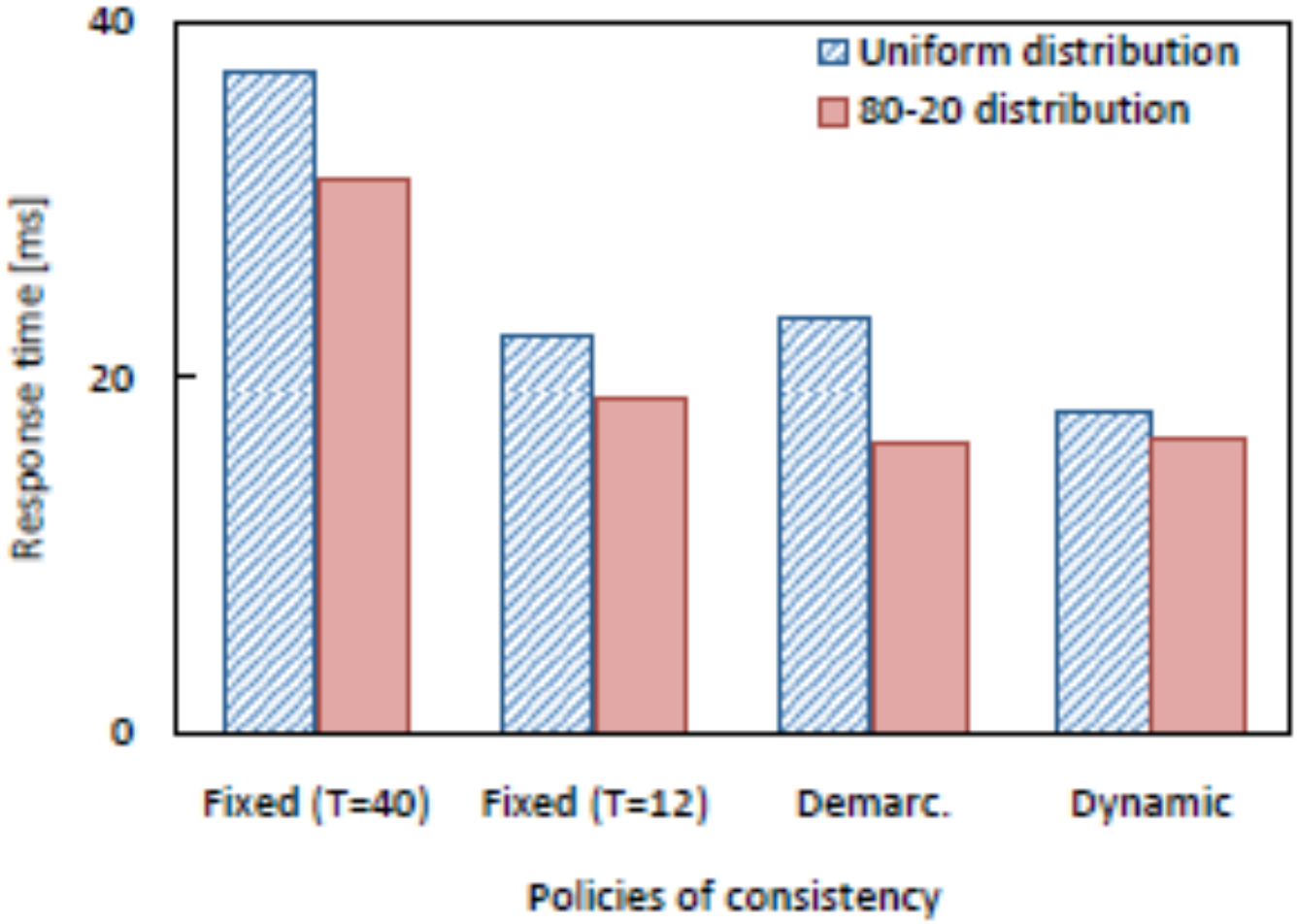
Response Time [ms]



Cost per TRX [$/1000]



Response Time [ms]



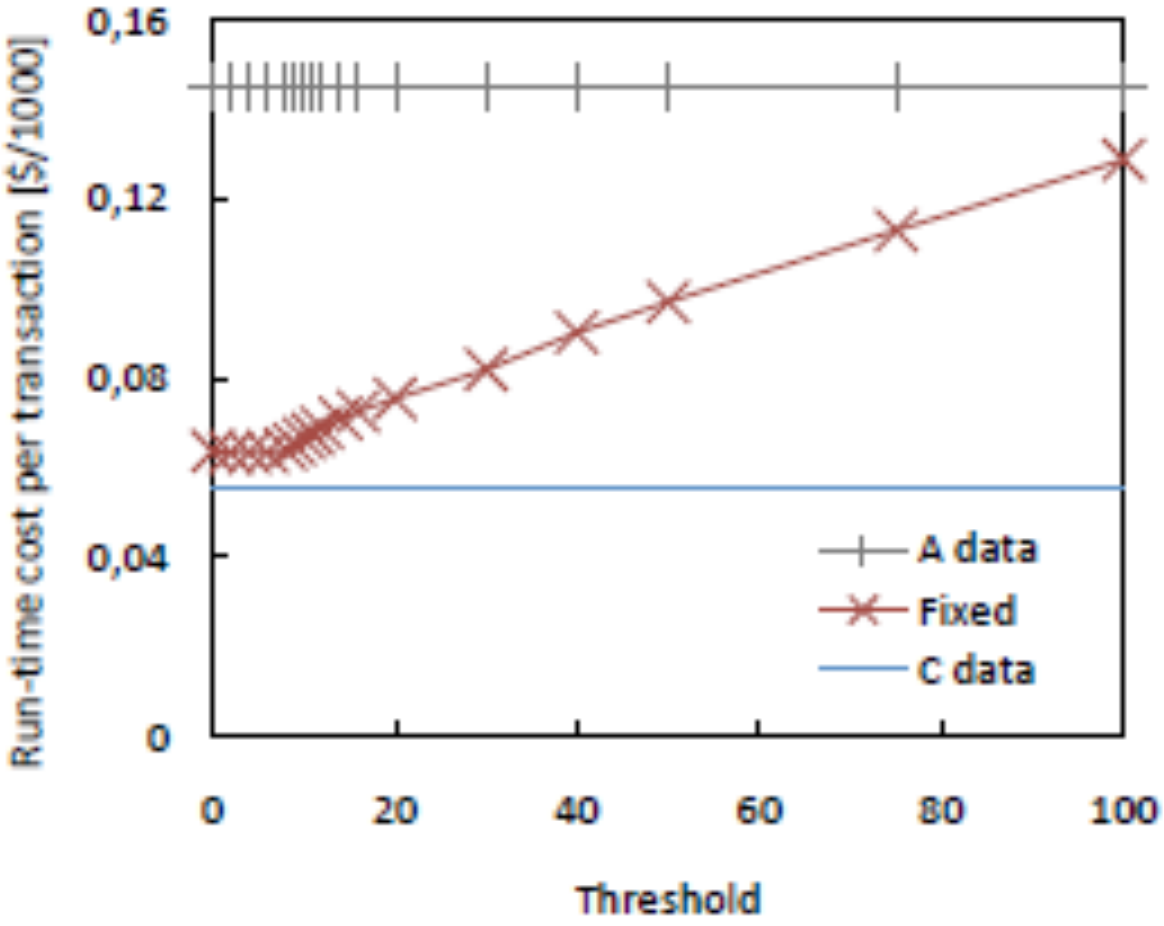


Figure 8: Runtime $, Vary threshold

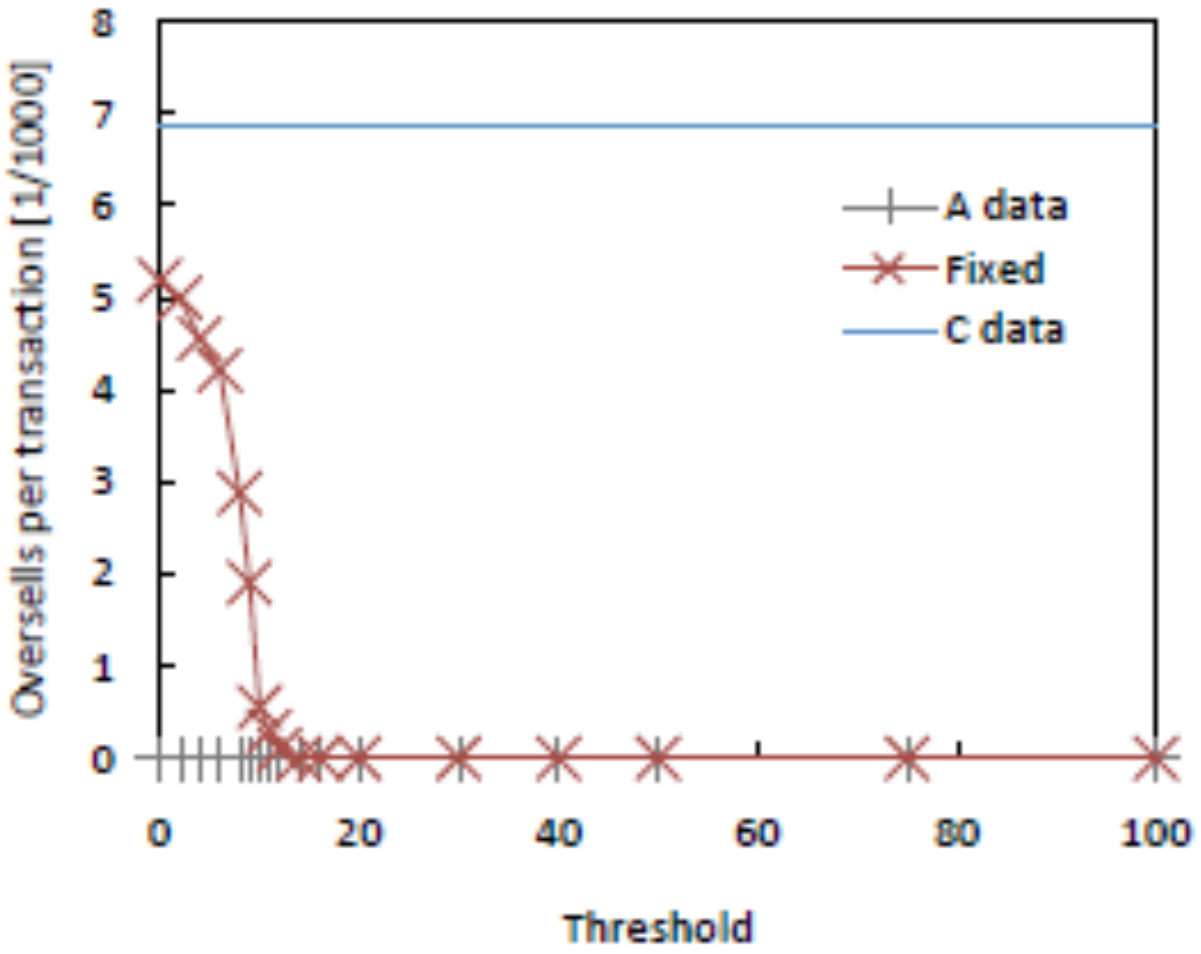


Figure 9: Oversells, Vary threshold

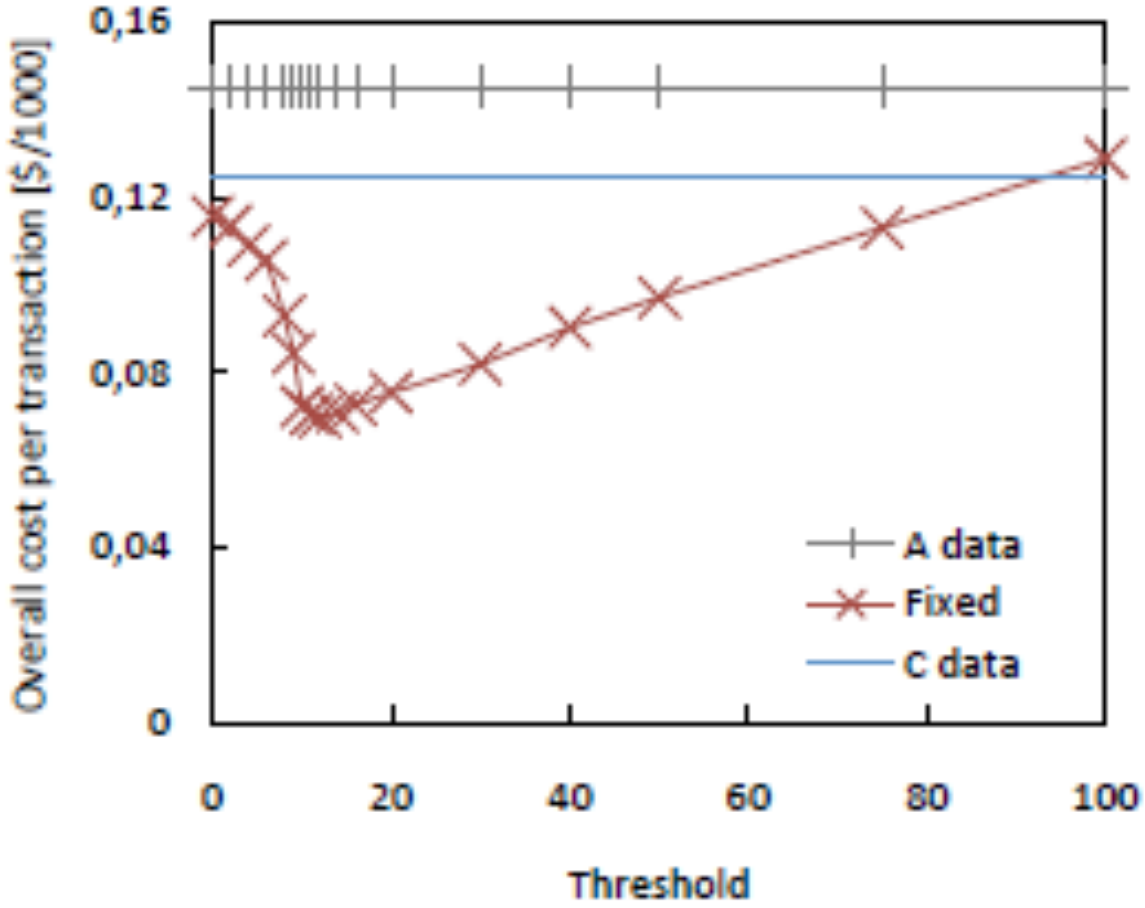


Figure 10: Overall $, Vary threshold

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