Consistency in Motion

Chris Douglas

A Familiar Problem

- Mixed workload
  - Small updates
  - Big analytic queries
  - Serializability

Berkeley

Concrete Scenario

```
BEGIN;

UPDATE Inventory I
  SET price = '700.99'
  WHERE I.product_id
     = 1234567;

END;
```

T1

```
BEGIN;

UPDATE Inventory I
  SET price = price * 1.02
  WHERE I.supplier = Apple

END;
```

T2

Storage Manager

BEGIN;

$R_1(x)$;
$W_1(y)$;

END;

T1

BEGIN;

$R_2(x)$;
$R_2(y)$;
…
$W_2(x)$;
$W_2(y)$;
…
END;

T2

Berkeley

Storage Manager

BEGIN;

$R_1(x_0)$;
$W_1(y_1)$;

END;

T1

BEGIN;

$R_2(x_0)$;
$R_2(y_0)$;
…
$W_2(x_2)$;
$W_2(y_2)$;
…
END;

T2

Storage Manager

T1	T2
BEGIN;	BEGIN;
$R_1(x_0)$;	$R_2(x_0)$;
	$R_2(y_0)$;
	…
$W_1(y_1)$;	$W_2(x_2)$;
	$W_2(y_2)$;
	…
END;	END;

Storage Manager

T1	T2
BEGIN;	BEGIN;
	$R_2(x_0)$;
	$R_2(y_0)$;
	…
$R_1(x_2)$; ←	$W_2(x_2)$;
$W_1(y_1)$;	$W_2(y_2)$;
	…
END;	END;

(╯°□°)╯︵ ┻━┻

T1		T2
BEGIN;		BEGIN;
$R_1(x_0)$;	$W^{-1}(y_0)$	$R_2(x_0)$;
$W_1(y_1)$;	$W_1(y_1)$	$R_2(y_0)$;
		…
		$W_2(x_2)$;
		$W_2(y_2)$;
		…
END;		END;

Berkeley

(╯°□°)╯︵ ┻━┻

BEGIN;

$R_1(x_0)$;
$W_1(y_1)$;

END;

T1

$W^{-1}(y_0)$
$W_1(y_1)$

BEGIN;

$R_2(x_0)$;
$R_2(y_1)$;
...
$W_2(x_2)$;
$W_2(y_2)$;
...

END;

T2

Berkeley

Can we do this in general?

1. Invertible writes
   a. Is this possible?
   b. How do we track state?

$\mathbb{Z}$-sets

2. Fix all the reads, cheaply
   a. Is this possible?
   b. Track state and update *incrementally*?

Materialized view maintenance

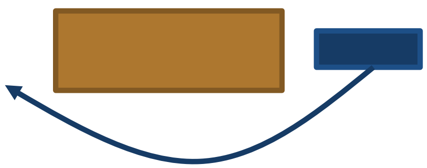
Agenda

- Motivation for reordering
- Inverse of a write
  - $\mathbb{Z}$-sets (🖹 Green, et al., 2009)
- Incremental *transaction* maintenance
  - IVM: Differential Dataflow, **DBSP** (🖹 Budiu, et al. 2022)
- Consistency in Motion: Reordering transactions
  - Dirty reads, aborts, etc.

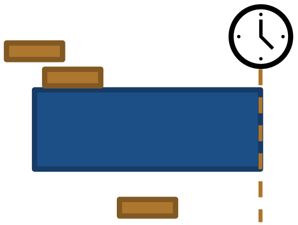
Berkeley

Reordering Transactions in Flight

Workload Heterogeneity
(HTAP)



Priority



Deadlines

Agenda

- Motivation for reordering
- Inverse of a write
  - $\mathbb{Z}$-sets (🗎 Green, et al., 2009)
- Incremental *transaction* maintenance
  - IVM: Differential Dataflow, **DBSP** (🗎 Budiu, et al. 2022)
- Consistency in Motion: Reordering transactions
  - Dirty reads, aborts, etc.

$\mathbb{Z}$-sets

Simple: weights on tuples

A $\mathbb{Z}$-set is a function $r: \tau \to \mathbb{Z}$

- τ the type of tuple
- finite support (finite rows)

Sets, Bags, sets of updates

- Positive weight: row(s) added
- Negative weight: row(s) deleted
- Zero: row not present

R

A	B	Weight
40	10	1
50	20	-1
20	10	2

$\mathbb{Z}$-sets

"Indifferent to ordering"

$(\mathbb{Z}, +, 0)$ is a commutative group

+ is associative, commutative

0 an identity element

inverse: $\forall e \in \mathbb{Z}, \exists e^{-1} : e + e^{-1} = 0$

R

A	B	Weight
40	10	1
50	20	-1
20	10	2

Berkeley

ℤ-Relations

- Every table has a (hidden) “weight” column
- Duplicates handled via the weight column
- Aggregates/outputs handled in the obvious way
  - Well-worn territory

Write and Write^{-1} in $\mathbb{Z}$-sets

- Write(t_{old}, t_{new}):
  - decrement(t_{old})
  - increment(t_{new})
- Write^{-1}(t_{old}, t_{new}):
  - decrement(t_{new})
  - increment(t_{old})

R

A	B	Weight
40	10	1
50	20	1
20	10	2

$+$

ΔR

A	B	Weight
50	20	-1
50	30	1

$=$

R

A	B	Weight
40	10	1
50	30	1
20	10	2

Write and Write^{-1} in $\mathbb{Z}$-sets

- Write(t_{old}, t_{new}):
  - decrement(t_{old})
  - increment(t_{new})
- Write^{-1}(t_{old}, t_{new}):
  - decrement(t_{new})
  - increment(t_{old})

R

A	B	Weight
40	10	1
50	20	1
20	10	2

+

ΔR

A	B	Weight
50	20	-1
50	30	1

=

R

A	B	Weight
40	10	1
50	30	1
20	10	2

- No blind writes
- Extra metadata

Read in ℤ-sets

- Read(t):
  - $\{t_1, \ldots t_n\}$ if weight(t) = n > 0
  - null if weight(t) <= 0

R

A	B	Weight
40	10	1
~~50~~	~~20~~	~~1~~
20	10	2

R

A	B	Weight
40	10	1
20	10	2

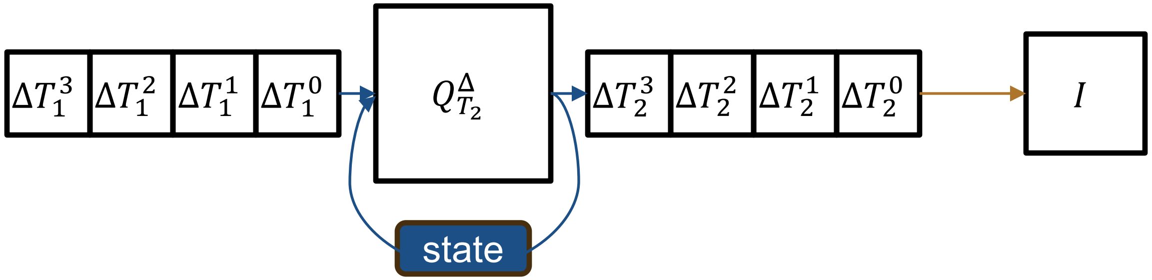
Agenda

- Motivation for reordering
- Inverse of a write
  - $\mathbb{Z}$-sets (🗎 Green, et al., 2009)
- Incremental *transaction* maintenance
  - IVM: Differential Dataflow, **DBSP** (🗎 Budiu, et al. 2022)
- Consistency in Motion: Reordering transactions
  - Dirty reads, aborts, etc.

Berkeley

DBSP

- Succinct (4 operator) streaming language
- Inputs *and* outputs are deltas (composable)
- Algorithm to convert **arbitrary** DBSP programs (query plans) to **incremental** DBSP programs
- Works over any commutative group (ℤ-sets)

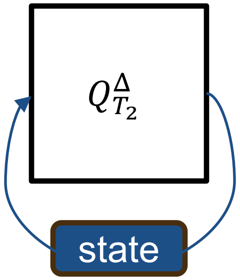


Berkeley

DBSP

- Succinct (4 operator) streaming language
- Inputs *and* outputs are deltas (composable)
- Algorithm to convert **arbitrary** DBSP programs (query plans) to **incremental** DBSP programs
- Works over any commutative group ($\mathbb{Z}$-sets)

Which inputs will produce a serializable execution of $Q_{T_2}^{\Delta}$?



DBSP Capabilities

- Streaming view maintenance system
  - Relational ($\sigma, \pi, \bowtie, \cup, -$)
  - Nested Relations (group-by, unnest, flatmap)
  - Aggregation
  - Streaming joins, window aggregates
  - Recursion ([non-]monotone, graph)
  - Stratified negation

SQL (Relational through Streaming joins, window aggregates)

Datalog (Recursion through Stratified negation)

Example: DBSP + Writes

- T_1: SELECT * FROM S INNER JOIN R WHERE A >= 30;
- T_2: UPDATE Table SET B = 10 WHERE A = 50;

- Assume T_1 ran before T_2; neither has committed

Berkeley

DBSP: Join

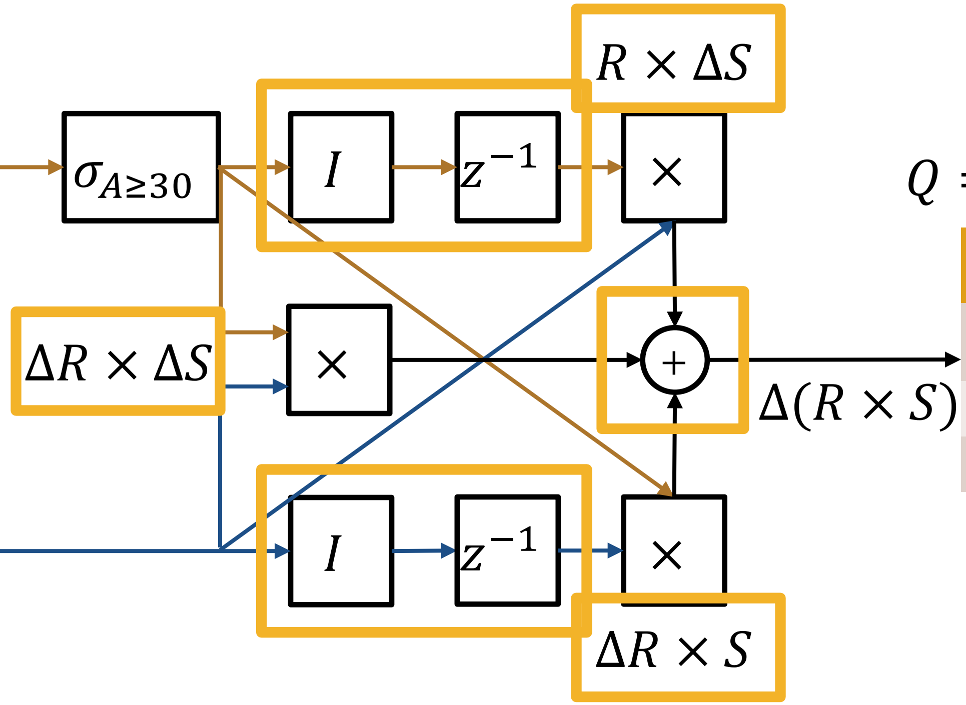
$$\Delta(R \times S) = \begin{matrix} R \times \Delta S + \\ \Delta R \times S + \\ \Delta R \times \Delta S \end{matrix}$$

R

A	B	Weight
40	10	1
50	20	2
20	30	1

S

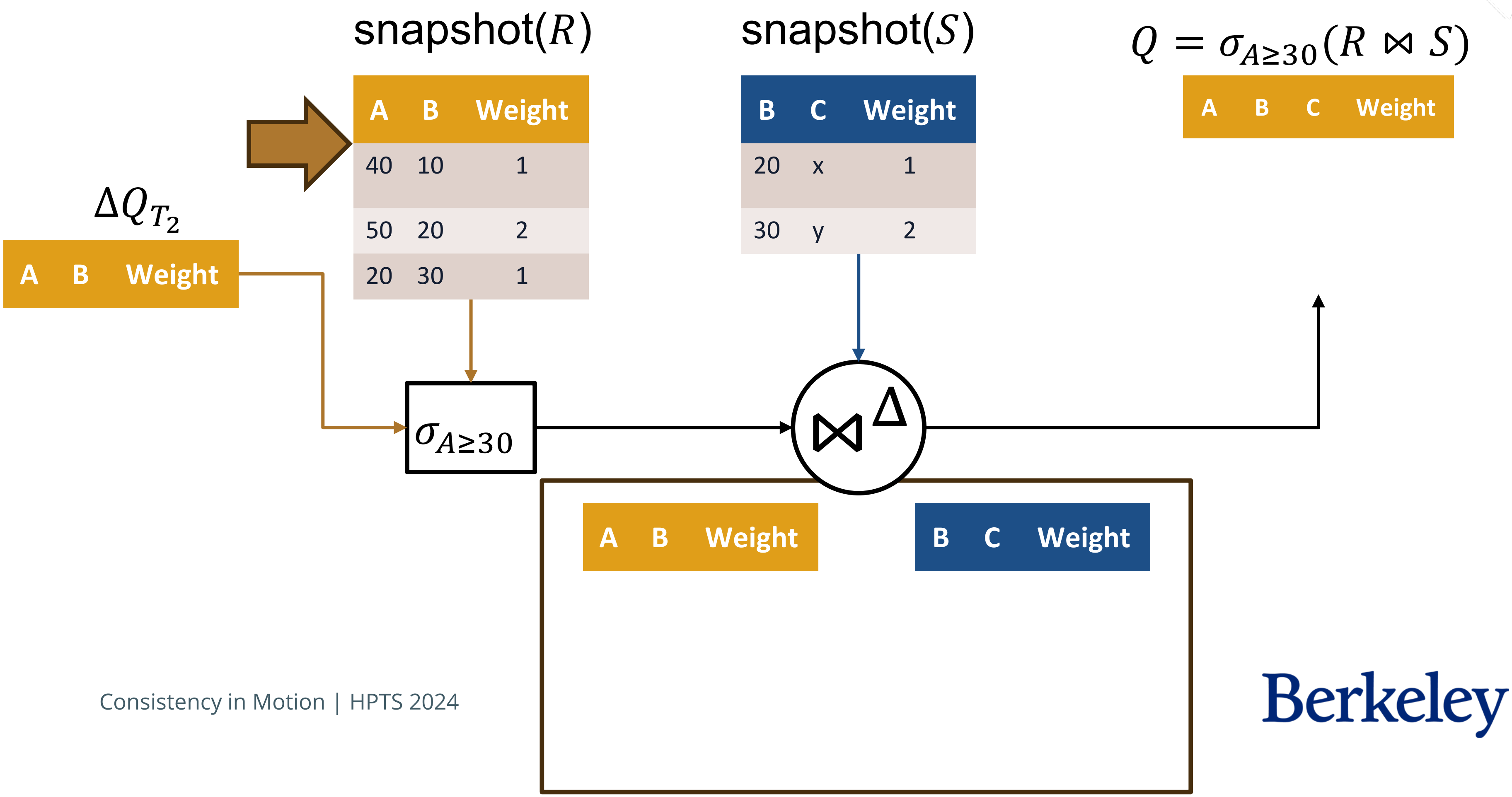
B	C	Weight
20	x	1
30	y	2



$Q = \sigma_{A\geq30}(R \bowtie S)$

A	B	C	Weight
50	20	x	2

DBSP: Join



snapshot(R)
snapshot(S)
$Q = \sigma_{A\geq30}(R \bowtie S)$
ΔQ_{T_2}
A B Weight
A B Weight
40 10 1
50 20 2
20 30 1
B C Weight
20 x 1
30 y 2
A B C Weight
$\sigma_{A\geq30}$
$\bowtie^{\Delta}$
A B Weight
B C Weight

DBSP: Join

ΔQ_{T_2}

A	B	Weight
50	20	-1
50	10	1

snapshot(R)

A	B	Weight
40	10	1
50	20	2
20	30	1

snapshot(S)

B	C	Weight
20	x	1
30	y	2

$Q = \sigma_{A\geq 30}(R \bowtie S)$

A	B	C	Weight

$\sigma_{A\geq 30}$

$\bowtie^{\Delta}$

A	B	Weight
40	10	1

B	C	Weight
20	x	1
30	y	2

Berkeley

DBSP: Join

ΔQ_{T_2}

A	B	Weight
50	20	-1
50	10	1

snapshot(R)

A	B	Weight
40	10	1
50	20	2
20	30	1

snapshot(S)

B	C	Weight
20	x	1
30	y	2

$Q = \sigma_{A \geq 30}(R \bowtie S)$

A	B	C	Weight
50	20	x	-1

$\sigma_{A \geq 30}$

$\bowtie^{\Delta}$

A	B	Weight
40	10	1
50	20	-1
50	10	1

B	C	Weight
20	x	1
30	y	2

Berkeley

DBSP: Join

ΔQ_{T_2}

A	B	Weight
50	20	-1
50	10	1

snapshot(R)

A	B	Weight
40	10	1
50	20	2
20	30	1

snapshot(S)

B	C	Weight
20	x	1
30	y	2

$Q = \sigma_{A \geq 30}(R \bowtie S)$

A	B	C	Weight
50	20	x	-1

$\sigma_{A \geq 30}$

$\bowtie^{\Delta}$

A	B	Weight
40	10	1
50	20	-1
50	10	1

B	C	Weight
20	x	1
30	y	2

Berkeley

DBSP: Join

ΔQ_{T_2}

A	B	Weight
50	20	-1
50	10	1
20	30	-1
50	30	1

snapshot(R)

A	B	Weight
40	10	1
50	20	2
20	30	1

snapshot(S)

B	C	Weight
20	x	1
30	y	2

$\sigma_{A\geq 30}$

$\bowtie^{\Delta}$

$Q = \sigma_{A\geq 30}(R \bowtie S)$

A	B	C	Weight
50	20	x	1
50	30	y	1

A	B	Weight
40	10	1
50	20	1
50	10	1
50	30	1

B	C	Weight
20	x	1
30	y	2

Berkeley

Agenda

- Motivation for reordering
- Inverse of a write
  - $\mathbb{Z}$-sets (🗎 Green, et al., 2009)
- Incremental *transaction* maintenance
  - IVM: Differential Dataflow, **DBSP** (🗎 Budiu, et al. 2022)
- Reordering transactions
  - Dirty reads, aborts, etc.

Berkeley

DBSP: Join

ΔQ_{T_2}

A	B	Weight
50	20	-1
50	10	1
20	30	-1
50	30	1

snapshot(R)

A	B	Weight
40	10	1
50	20	2
20	30	1

snapshot(S)

B	C	Weight
20	x	1
30	y	2

$Q = \sigma_{A \geq 30}(R \bowtie S)$

A	B	C	Weight
50	20	x	1
50	30	y	1

$\sigma_{A \geq 30}$

$\bowtie^{\Delta}$

A	B	Weight
40	10	1
50	20	1
50	10	1
50	30	1

B	C	Weight
20	x	1
30	y	2

Berkeley

DBSP: Join

snapshot(R)

A	B	Weight
40	10	1
50	20	2
20	30	1

snapshot(S)

B	C	Weight
20	x	1
30	y	2

$Q = \sigma_{A\geq 30}(R \bowtie S)$

A	B	C	Weight
50	20	x	2
50	30	y	0

ΔQ_{T_2}

A	B	Weight
50	20	1
50	10	-1
20	30	1
50	30	-1

$\sigma_{A\geq 30}$

$\bowtie^{\Delta}$

A	B	Weight
40	10	1
50	20	2
50	10	0
50	30	0

B	C	Weight
20	x	1
30	y	2

Berkeley

Skepticism

- Delete anomalies
  - T_1, T_2 read the same snapshot, delete the same record
- Bookkeeping overheads
  - No blind writes
  - T_1, T_2 starting from different snapshots
- DBSP compilation is *expensive*
- Only beneficial if it improves goodput/makespan
  - More work per query, need to make that up
- Too complex in practice?
- Not the target for DBSP. Other tools?

Berkeley

Wild Optimism

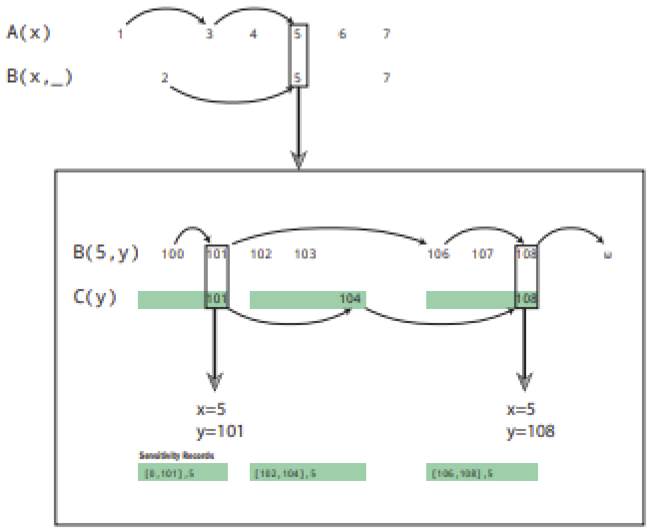
- **Cost-aware reordering**
  - If an update causes a large ΔT_2 for $Q_{T_2}^{\Delta}$, reject it
  - Queries with **deadlines** can become increasingly restrictive
- **Speculation** for distributed concurrency control
  - Local sequencer speculatively orders transactions
  - Repair batch with transactions from remote replicas
- **Undo dependencies** instead of waiting
  - Mispredictions can be "fixed"
  - Speculatively make dep. durable, then undo + commit
- **Deferred commit** of transactions that violate constraints
  - Outcome-oriented schedule optimization
  - Reorder transactions ahead until integrity constraint holds

Related work: LogicBlox

- LogiQL (Datalog extension)
- Worst-case optimal join
  - Leapfrog Trie Join
- Incremental maintenance
- Full serializability

Veldhuizen, Todd L. "Incremental maintenance for leapfrog triejoin." arXiv preprint arXiv:1303.5313 (2013).

Veldhuizen, Todd L. "Transaction repair: Full serializability without locks." arXiv preprint arXiv:1403.5645 (2014).

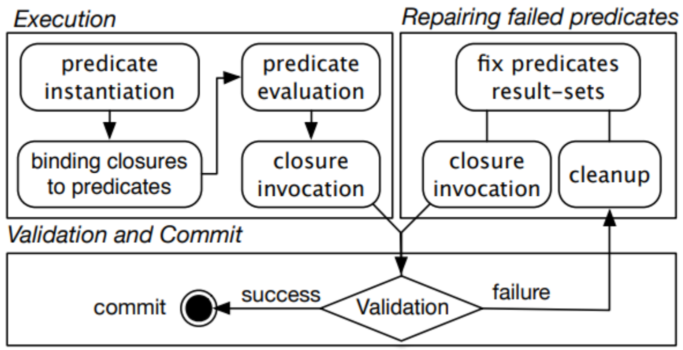


(a) Example of Leapfrog Triejoin in operation (see text).

Berkeley

Related work: Transaction Repair

- Re-execute only conflicting subsets of txn



Dashti, Mohammad, Sachin Basil John, Amir Shaikhha, and Christoph Koch. "Transaction repair for multi-version concurrency control." In Proceedings of the 2017 ACM International Conference on Management of Data, pp. 235-250. 2017.

Burke, Matthew, Florian Suri-Payer, Jeffrey Helt, Lorenzo Alvisi, and Natacha Crooks. "Morty: Scaling Concurrency Control with Re-Execution." In Proceedings of the Eighteenth European Conference on Computer Systems, pp. 687-702. 2023.

Dong, Zhiyuan, Zhaoguo Wang, Xiaodong Zhang, Xian Xu, Changgeng Zhao, Haibo Chen, Aurojit Panda, and Jinyang Li. "Fine-Grained Re-Execution for Efficient Batched Commit of Distributed Transactions." Proceedings of the VLDB Endowment 16, no. 8 (2023): 1930-1943.

Questions?

- Transaction reordering by incremental maintenance
  - Related work?
  - Possibly relevant workloads?
  - Confounding problems?

chris_douglas@berkeley.edu

Thank you!
Mihai Budiu, Val Tannen, Tiemo Bang, Conor Power,
Natacha Crooks, Joe Hellerstein

Berkeley

References

- Budiu, Mihai, Tej Chajed, Frank McSherry, Leonid Ryzhyk, and Val Tannen. "DBSP: Automatic Incremental View Maintenance for Rich Query Languages." Proceedings of the VLDB Endowment 16, no. 7 (2023): 1601-1614.
- Green, Todd J., Zachary G. Ives, and Val Tannen. "Reconcilable differences." In *Proceedings of the 12th International Conference on Database Theory*, pp. 212-224. 2009.
- Veldhuizen, Todd L. "Incremental maintenance for leapfrog triejoin." arXiv preprint arXiv:1303.5313 (2013).
- Veldhuizen, Todd L. "Transaction repair: Full serializability without locks." arXiv preprint arXiv:1403.5645 (2014).
- Burke, Matthew, Florian Suri-Payer, Jeffrey Helt, Lorenzo Alvisi, and Natacha Crooks. "Morty: Scaling Concurrency Control with Re-Execution." In Proceedings of the Eighteenth European Conference on Computer Systems, pp. 687-702. 2023.
- Dong, Zhiyuan, Zhaoguo Wang, Xiaodong Zhang, Xian Xu, Changgeng Zhao, Haibo Chen, Aurojit Panda, and Jinyang Li. "Fine-Grained Re-Execution for Efficient Batched Commit of Distributed Transactions." Proceedings of the VLDB Endowment 16, no. 8 (2023): 1930-1943.
- Dashti, Mohammad, Sachin Basil John, Amir Shaikhha, and Christoph Koch. "Transaction repair for multi-version concurrency control." In Proceedings of the 2017 ACM International Conference on Management of Data, pp. 235-250. 2017.

Berkeley

DBSP ⊆ Differential Dataflow

DBSP (so far)

- Synchronous
  - Unique predecessor
- Choose system/event time
  - other is “regular data”
- Maintains up-to-date state

Differential Dataflow

- Partial order
  - Captures causality
- Out-of-order
  - Patch the present with past events
- Complex state
  - Möbius inversion
  - see: “Foundations of Differential Dataflow”

Berkeley