Lazy Evaluation of Transactions in Database Systems

Jose M. Faleiro♦, Alexander Thomson★, Daniel J. Abadi♦

♦Yale University, ★Google
Paper “X” looks relevant, can you add it to related work?
Just read through the whole thing and added to related work. Our work is more ACIDic.
In reality…

• Read only abstract to give a sensible response

• Keep “sticky” note on desk to later read paper and write related work.
Read paper X
Add to related work

Read paper Y
Add to related work

Read paper Z
Add to related work
Hey, can you check-in your additions to related work? I can’t see your changes…
Read paper X
Add to related work

Read paper Y
Add to related work

Read paper Z
Add to related work

!!!
• Previously said I did all the work. Can’t feign ignorance.
• Use information on stickies to read papers and finish writing related work.
Lazy Database Systems

• Promise to commit/abort while only partially executing txns

• Keep promises while maintaining ACID and serializability
Lying Considered Harmful Useful

- Flexibility to execute transactions when most favorable
  - Cache Locality
  - Load Balancing
  - Contention Reduction
Can’t Keep Lying Forever

Must satisfy “external” reads
Talk Outline

- Lazy Database Design
- Benefits of Lazy Execution
- Experimental Evaluation
- Conclusion
Customer Orders Product

Update product inventory

Compute discounted price

Update customer bill
Customer Orders Product

**Update product inventory**

- Compute discounted price

- Update customer bill

```
Read Product record

Product count -= 1

If Product count < 0:
    ABORT()
```
Customer Orders Product

Update product inventory

Compute discounted price

Update customer bill

Read Product record
Product count -= 1
If Product count < 0:
ABORT()

Read daily discount
Compute discounted price
**Customer Orders Product**

Update product inventory

Compute discounted price

Update customer bill

```
Read Product record
Product count -= 1
If Product count < 0:
    ABORT()
Read daily discount
Compute discounted price

Read customer’s monthly bill record
Customer’s monthly bill += Discounted price

COMMIT()
```
We Ask the Question

Can we: return commit/abort promise … without executing to completion…

while maintaining ACID and serializability?

Read Product record
Product count -= 1
If Product count < 0:
   ABORT()
Read daily discount
Compute discounted price
Read customer’s monthly bill record
Customer’s monthly bill += Discounted price
COMMIT()
Our Solution

• Split up transactions into two parts
  – Give users the illusion that execution is atomic

• First part produces commit/abort promise
  – Executes immediately

• Second part does everything else
  – Execution is deferred
How to Split a Transaction?

Read Product record

Product count -= 1

If Product count < 0:
   \textcolor{red}{ABORT()}\textcolor{red}{(})

Read daily discount

Compute discounted price

Read customer’s monthly bill record

Customer’s monthly bill += Discounted price

\textcolor{green}{COMMIT()}
Splitting a Transaction

Abort due to txn logic

Read Product record
Product count -= 1
If Product count < 0:
ABORT()
Read daily discount
Compute discounted price
Read customer’s monthly bill record
Customer’s monthly bill += Discounted price
COMMIT()
Splitting a Transaction

Read Product record

Product count -= 1

If Product count < 0:
    ABORT()

Read daily discount

Compute discounted price

Read customer's monthly bill record

Customer's monthly bill += Discounted price

COMMIT()

Return

commit

promise
Splitting a Transaction

Read Product record

Product count -= 1

If Product count < 0:
   ABORT()

Read daily discount

Compute discounted price

Read customer’s monthly bill record

Customer’s monthly bill += Discounted price

COMMIT()
Splitting a Transaction

Read Product record

Product count -= 1

If Product count < 0:
   ABORT()

COMMIT_PROMISE()

Read daily discount

Compute discounted price

Read customer’s monthly bill record

Customer’s monthly bill += Product price
Splitting a Transaction

**Execute immediately**

- Read Product record
- Product count -= 1
- If Product count < 0:
  - ABORT()
  - COMMIT_PROMISE()

**Defer execution**

- Read daily discount
- Compute discounted price
- Read customer’s monthly bill record
- Customer’s monthly bill += Product price
Dealing with Deferred Logic

• Don’t write out actual record values

• **Blindly** insert placeholders corresponding to unexecuted deferred logic
  – Similar to stickies on my desk as reminders
Dealing with Deferred Logic

Monthly Bill

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>$17.05</td>
</tr>
<tr>
<td>Mary</td>
<td></td>
</tr>
<tr>
<td>David</td>
<td>$300</td>
</tr>
</tbody>
</table>

Read Tuesday’s discount
Compute discounted price
Read customer’s monthly bill record
Mary’s monthly bill += Product price
Processing External Reads

• External read may depend on a deferred transaction

• Deferred transaction itself may have dependencies
Dealing with Deferred Logic

Monthly Bill

<table>
<thead>
<tr>
<th>Name</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>$17.05</td>
</tr>
<tr>
<td>Mary</td>
<td></td>
</tr>
<tr>
<td>David</td>
<td>$300</td>
</tr>
</tbody>
</table>

Read Tuesday’s discount
Compute discounted price
Read customer’s monthly bill record
Mary’s monthly bill += Product price

Daily Discount

<table>
<thead>
<tr>
<th>Day</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>5%</td>
</tr>
<tr>
<td>Tuesday</td>
<td></td>
</tr>
<tr>
<td>Friday</td>
<td>15%</td>
</tr>
</tbody>
</table>

Increase Tuesday’s discount by 10%
Dealing with Deferred Logic

Monthly Bill

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>$17.05</td>
</tr>
<tr>
<td>Mary</td>
<td></td>
</tr>
<tr>
<td>David</td>
<td>$300</td>
</tr>
</tbody>
</table>

Daily Discount

<table>
<thead>
<tr>
<th>Day</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>5%</td>
</tr>
<tr>
<td>Tuesday</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Friday</td>
<td>15%</td>
</tr>
</tbody>
</table>

Read Tuesday’s discount
Compute discounted price
Read customer’s monthly bill record
Mary’s monthly bill += Product price
Increase Tuesday’s discount by 10%
Dealing with Deferred Logic

Deferred pieces of logic are implicitly ordered.

Read Tuesday’s discount
Compute discounted price
Read customer’s monthly bill record
Mary’s monthly bill += Product price
Increase Tuesday’s discount by 10%
Dependency Graph of Transactions
Dependency Graph of Transactions

R: A
W: B

R: B
W: Y

R: X, Z
W: C
Dependency Graph of Transactions

- **Read (R):** A, B, X, Z
- **Write (W):** B, Y, C

Diagram:
- Node 1: R: A, W: B
- Node 2: R: B, W: Y

Arrows indicate dependency relationships between transactions.
Dependency Graph of Transactions

- Read: A, Write: B
- Read: B, Write: Y
- Read: Y, Write: X
- Read: X, Z, Write: C
- Read: Y, Write: Z
Dependency Graph of Transactions

Record Z?
Dependency Graph of Transactions

Record Z?

Execute last writer and its transitive closure
Large Transitive Closures

• The subset of the graph a record depends on may grow very large
  – High external read latency
Large Transitive Closures

• The subset of the graph a record depends on may grow very large
  – High external read latency
  – Incrementally process transactions in the background
Incremental Transaction Processing

T1, T2, T3, T4, T5
Incremental Transaction Processing

Bound the maximum depth of the dependency graph
Incremental Transaction Processing

Bound the maximum depth of the dependency graph
Incremental Transaction Processing

Tradeoff:

• Higher bound means larger batches (good for cache locality)

• Larger batches mean increased external read latency
Talk Outline

• Lazy Database Design

• Benefits of Lazy Execution

• Experimental Evaluation

• Conclusion
Benefits of Lazy Transactions

- Data Cache/Buffer Pool Locality
- Temporal Load Balancing
- Reduced Contention
Execute immediately

Read Product1 record

Product1 count -= 1

If Product1 count < 0:
   ABORT()

COMMIT_PROMISE()

Read Tuesday’s discount

Compute discounted price

Read Mary’s monthly bill record

Mary’s monthly bill += Product price
Read Tuesday’s discount

Compute discounted price

Read Mary’s monthly bill record

Mary’s monthly bill +=

Product price

Deferred
Data Cache/Buffer Pool Locality

Deferred

Read Mary’s monthly bill record
Mary’s monthly bill += Product1 price
Insert a record into Orders table

Read Product2 record
Product2 count -= 1
If Product2 count < 0:
   ABORT()
   COMMIT_PROMISE()

Read Monday’s discount
Compute discounted price
Read Mary’s monthly bill record
Mary’s monthly bill += Product price
Data Cache/Buffer Pool Locality

Read Mary’s monthly bill record

Read Monday’s discount

Compute discounted price

Read Mary’s monthly bill record

Mary’s monthly bill += Product price

Deferred
Deferred
Data Cache/Buffer Pool Locality

Read Monday’s discount
Compute discounted price
Read Mary’s monthly bill record
Mary’s monthly bill += Product2 price

Both update the same bill

Deferred

Read Tuesday’s discount
Compute discounted price
Read Mary’s monthly bill record
Mary’s monthly bill += Product1 price
Data Cache/Buffer Pool Locality

Deferred

Read Monday's discount
Compute discounted price
Read Mary's monthly bill record
Mary's monthly bill += Product2 price

Read Tuesday's discount
Compute discounted price
Read Mary's monthly bill record
Mary's monthly bill += Product1 price

Bring Mary's bill record into cache just once
Temporal Load Balancing

Read Product record

Product count -= 1

If Product count < 0:
   ABORT()

Read daily discount

Compute discounted price

Read Customer’s monthly bill record

Customer’s monthly bill +=
   Product price

COMMIT()

---

Read Product record

Product count -= 1

If Product count < 0:
   ABORT()

COMMIT.Promise()

Read daily discount

Compute discounted price

Read Customer’s monthly bill record

Customer’s monthly bill +=
   Product price
Temporal Load Balancing

Read Product record
Product count -= 1
If Product count < 0:
   ABORT()
Read daily discount
Compute discounted price
Read Customer’s monthly bill record
Customer’s monthly bill += Product price
COMMIT()

Arrows represent time required to immediately process transactions
Temporal Load Balancing

More commit decisions per unit time

Eager

Lazy
Reduced Contention

Popular item

Read iThing record

iThing count -= 1

If iThing count < 0:
    ABORT()

Read daily discount

Compute discounted price

Read Fanboy1’s monthly bill record

Fanboy1’s monthly bill += Discounted price

COMMIT()
Lock must be held for this period.

Reduced Contention

Read iThing record

iThing count -= 1

If iThing count < 0:
    ABORT()

Read daily discount

Compute discounted price

Read Fanboy1’s monthly bill record

Fanboy1’s monthly bill += Discounted price

COMMIT()
Reduced Contention

Hold contended lock for less time

Read daily discount

Compute discounted price

Read Fanboy1’s monthly bill record

Fanboy1’s monthly bill += Discounted price

Read iThing record

iThing count -= 1

If iThing count < 0:
  ABORT()

COMMIT_PROMISE()
Talk Outline

• Lazy Database Design

• Benefits of Lazy Execution

• Experimental Evaluation

• Conclusion
Experimental Setup

• All data in memory

• Single machine

• 8 cores
  – Lazy: 1 commit/abort + 7 deferred logic
  – Eager: 8 cores
Experimental Evaluation

- Cache Locality
- External Read Latency
- Temporal Load Balancing
- Contention Reduction
Cache- Locality

- “Cliquey” workload
- Conflicting transactions conflict on several records

![Diagram showing key space and R/W sets for transactions A, B, and C]
Cache-Locality

Throughput (tps) vs. Path Length Bound

- Lazy
- Eager
External Read Latency CDF

Fraction vs. Latency in $\mu$ seconds

- **Lazy Path 20**
- **Lazy Path 100**
- **Eager**
Temporal Load Balancing

• Lazy and eager systems have comparable throughput
Temporal Load Balancing

[Graph showing temporal load balancing over time with different load balancer strategies: Input, Lazy, Eager, Commit Decision. The graph illustrates the number of input, processed, and successful transactions over time.]
Conclusions

• Lazily evaluating txns has its benefits
  – Cache Locality
  – Temporal Load Balancing
  – Contention Reduction

To see the rest of our conclusions, read the paper
Microbenchmark Throughput

Throughput (txns/sec)

No Spatial Locality

Lazy

Eager

With Spatial Locality

Path Length Bound
Microbenchmark Latency

- Lazy Path 20
- Lazy Path 60
- Lazy Path 100
- Eager

No Spatial Locality

With Spatial Locality

Fraction of Transactions vs. Latency in μ seconds