EVERYTHING I KNOW ABOUT FAST DATABASES I LEARNED AT THE DOG TRACK

@andy_pavlo
VOTER BENCHMARK
Japanese “American Idol”

TRANSACTION
1. Check whether user has already voted.
2. Insert new vote entry.
3. Update vote count for contestant.
VOTER BENCHMARK

Japanese “American Idol”

<table>
<thead>
<tr>
<th>CPU CORES</th>
<th>TXN/SEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
<tr>
<td>2</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
<tr>
<td>3</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
<tr>
<td>4</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
<tr>
<td>5</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
<tr>
<td>6</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
<tr>
<td>7</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
<tr>
<td>8</td>
<td>MySQL</td>
</tr>
<tr>
<td></td>
<td>Postgres</td>
</tr>
</tbody>
</table>
TRADITIONAL DBMS
Measured CPU Cycles

- BUFFER POOL: 30%
- LOCKING: 28%
- RECOVERY: 30%
- REAL WORK: 12%

OLTP THROUGH THE LOOKING GLASS, AND WHAT WE FOUND THERE
CAN YOU SCALE UP WITHOUT GIVING UP TRANSACTIONS?
Fast  Repetitive  Small
Optimization

USE A LIGHTWEIGHT SYSTEM DESIGNED FOR OLTP TRANSACTIONS.

H-STORE: A HIGH-PERFORMANCE, DISTRIBUTED MAIN MEMORY TRANSACTION PROCESSING SYSTEM
DISK ORIENTED
MAIN MEMORY STORAGE

CONCURRENT EXECUTION
SERIAL EXECUTION

HEAVYWEIGHT RECOVERY
COMPACT LOGGING
H-Store

PARTITIONS

SINGLE-THREADED EXECUTION ENGINES
H-Store

Transaction Result

Application

SNAPSHOTS
DISTRIBUTED TRANSACTIONS
TPC-C BENCHMARK
8 Cores per Node
10% Distributed Transactions

H-Store
DISTRIBUTED TRANSACTION
KNOW WHAT THINGS WILL DO BEFORE THEY START RUNNING
Optimization

USE MODELS TO PREDICT TRANSACTION BEHAVIOR BEFORE EXECUTION.

ON PREDICTIVE MODELING FOR OPTIMIZING TRANSACTION EXECUTION IN PARALLEL OLTP SYSTEMS
SELECT * FROM WAREHOUSE WHERE W_ID = 10;
SELECT * FROM DISTRICT WHERE D_W_ID = 10 AND D_ID = 9;
INSERT INTO ORDERS (O_W_ID, O_D_ID, O_C_ID,...) VALUES (10, 9, 12345,...);
DISTRIBUTED TRANSACTION

begin
QueryX
  Counter: 0
  Partitions: {0}
  Previous: ∅

QueryY
  Counter: 0
  Partitions: {0, 1, 2, 3}
  Previous: {0}

QueryX
  Counter: 1
  Partitions: {0}
  Previous: {0, 1, 2, 3}

QueryZ
  Counter: 0
  Partitions: {1}
  Previous: {0, 1, 3, 4}

QueryZ
  Counter: 1
  Partitions: {1}
  Previous: {0, 1, 3, 4}

commit
DISTRIBUTED TRANSACTION

begin

QueryX
Counter: 0
Partitions: { 0 }
Previous: ∅

QueryY
Counter: 0
Partitions: { 0,1,2,3 }
Previous: { 0 }

QueryX
Counter: 1
Partitions: { 0 }
Previous: { 0,1,2,3 }

QueryZ
Counter: 0
Partitions: { 1 }
Previous: { 0,1,3,4 }

QueryZ
Counter: 1
Partitions: { 1 }
Previous: { 0,1,3,4 }

commit

Application
TPC-C BENCHMARK
8 Cores per Node

Naïve  Houdini

Optimal

2x

TXN/SEC

CPU CORES

0 5,000 10,000 15,000 20,000 25,000

8 16 32
TPC-C BENCHMARK
8 Cores per Node

TXN/SEC vs CPU CORES

Naïve vs Houdini
DISTRIBUTED TRANSACTION

Application
DO SOMETHING
USEFUL
WHENEVER YOU ARE IDLE
Optimization

SPECULATIVELY EXECUTE TRANSACTIONS WHEN THE SYSTEM IS STALLED.
SERIALIZABLE SCHEDULE

Distributed Transaction

Single-Partition Transaction

Single-Partition Transaction
SERIALIZABLE SCHEDULE

Distributed Transaction

Speculative Transaction

Speculative Transaction
Speculation Candidate:

Distributed Transaction:
Hermes

Transaction Queue
STORED PROCEDURE

VoteCount:

```
SELECT COUNT(*)
FROM votes
WHERE phone_num = ?;
```

InsertVote:

```
INSERT INTO votes
VALUES (?, ?, ?, ?);
```

```
run (phoneNumber, contestantId, currentTime) {
    result = execute (VoteCount, phoneNumber);
    if (result > MAX_VOTES) {
        return (ERROR);
    }
    execute (InsertVote, phoneNumber, contestantId, currentTime);
    return (SUCCESS);
}
```
Input Parameters:
- \( w\_id = 0 \)
- \( i\_w\_ids = [1, 0] \)
- \( i\_ids = [1001, 1002] \)

GetWarehouse:
```
SELECT * FROM WAREHOUSE WHERE W_ID = ?
```
Input Parameters:

- \( w\_id = 0 \)
- \( i\_w\_ids = [1,0] \)
- \( i\_ids = [1001, 1002] \)

CheckStock:

```
SELECT S\_QTY FROM STOCK
WHERE S\_W\_ID = ?
AND S\_I\_ID = ?;
```
TPC-C BENCHMARK

8 Cores per Node

Graph showing performance metrics across different CPU cores for Naïve, Houdini, and Hermes. The x-axis represents CPU cores (8, 16, 32), and the y-axis represents TXN/SEC ranging from 0 to 80,000. The graph indicates an upward trend for all three metrics with Naïve having the lowest performance, followed by Houdini, and Hermes showing the highest performance. The term “OPTIMAL” is used to denote the peak performance point.
The more you know, the faster you can go.
REAL-WORLD IMPACT

• On-line Voting Applications.
• On-line Gaming & Gambling.
• Network Traffic Monitoring.
• High-Frequency Trading.
OPTIMIZE SYSTEM FOR TRANSACTIONS

PREDICT TRANSACTION BEHAVIOR

SAFELY INTERLEAVE TRANSACTIONS
FUTURE
WORK
WORKLOAD EXPANSION

- Stream Processing.
- Non-Partitionable Workloads.
- Real-time Analytics.
  - Hybrid Storage Models.
  - Relaxed Consistency.
IMPROVING SCALABILITY

- Elastic Deployments.
- New Hardware:
  - Non-Volatile Memory.
  - Many-Core Architectures.
- Anti-Caching.
ANTI-CACHING

Application

Anti-Cache
ANTI-CACHING

Application

Anti-Cache
HOW TO SCALE UP WITHOUT GIVING UP TRANSACTIONS.
H-Store
hstore.cs.cs.brown.edu
@andy_pavlo