Magical Parallel OLTP Databases

Andy Pavlo
Lebron is going to Miami!
The McRib will be back!
Michael Jackson is in trouble!
On-Line Transaction Processing
H-Store
**H-Store Partitioning**

Tables → Partitions

- WAREHOUSE
- DISTRICT
- CUSTOMER
- ORDERS
- ORDER_ITEM
- STOCK
- ITEM

Partitions P1, P2, P3, P4, P5

Replicated
NewOrder extends StoredProcedure {

Query GetWarehouse = "SELECT * FROM WAREHOUSE WHERE W_ID = ?";
Query CheckStock = "SELECT S_QTY FROM STOCK
    WHERE S_W_ID = ? AND S_I_ID = ?";
Query InsertOrder = "INSERT INTO ORDERS VALUES (?,?)";
Query InsertOrdLine = "INSERT INTO ORDER_LINE VALUES (?,?,?,?,?)";
Query UpdateStock = "UPDATE STOCK SET S_QTY = S_QTY - ?
    WHERE S_W_ID = ? AND S_I_ID = ?";

run(int w_id, int i_ids[], int i_w_ids[], int i_qtys[]) {
    queueSQL(GetWarehouse, w_id);
    for (int i = 0; i < i_ids.length; i++)
        queueSQL(CheckStock, i_w_ids[i], i_ids[i]);
    Result r[] = executeBatch();
    int o_id = r[0].get("W_NEXT_O_ID") + 1;
    queueSQL(InsertOrder, w_id, o_id);
    for (int i = 0; i < r.length; i++) {
        if (r[i+1].get("S_QTY") < i_qtys[i]) abort();
        queueSQL(InsertOrdLine, w_id, o_id, i_ids[i], i_qtys[i]);
        queueSQL(UpdateStock, i_qtys[i], i_w_ids[i], i_ids[i]);
    }
    return (executeBatch() != null);
}
This transaction will execute 4 queries on partitions 1, 3, and 6!
Optimization #1
Optimization #2

Client Application

Database Node

Core

Tax Coordinator

Execution Engine

Partition Data

Main Memory

Partition Data

Partition Data

Partition Data

Partition Data

Partition Data

Partition Data

Partition Data

Partition Data

Partition Data

Partition Data

Partition Data
Optimization #2
class NewOrder extends StoredProcedure {
    Query GetWarehouse = "SELECT * FROM WAREHOUSE WHERE W_ID = ?";
    Query CheckStock = "SELECT S_QTY FROM STOCK
                        WHERE S_W_ID = ? AND S_I_ID = ?";
    Query InsertOrder = "INSERT INTO ORDERS VALUES (?,?)";
    Query InsertOrderLine = "INSERT INTO ORDER_LINE VALUES (?,?,?,?)";
    Query UpdateStock = "UPDATE STOCK SET S_QTY = S_QTY - ?
                         WHERE S_W_ID = ? AND S_I_ID = ?";

    int run(int w_id, int i_ids[], int i_w_ids[], int i_qtys[]) {
        queueSQL(GetWarehouse, w_id);
        for (int i = 0; i < i_ids.length; i++)
            queueSQL(CheckStock, i_w_ids[i], i_ids[i]);
        Result r[] = executeBatch();
        int o_id = r[0].get("W_NEXT_O_ID") + 1;
        queueSQL(InsertOrder, w_id, o_id);
        for (int i = 0; i < r.length; i++) {
            if (r[i+1].get("S_QTY") < i_qtys[i]) abort();
            queueSQL(InsertOrderLine, w_id, o_id, i_ids[i], i_qtys[i]);
            queueSQL(UpdateStock, i_qtys[i], i_w_ids[i], i_ids[i]);
        }
        return (executeBatch() != null);
    }
}
class NewOrder extends StoredProcedure {
    Query GetWarehouse = "SELECT * FROM WAREHOUSE WHERE W_ID = ?";
    Query CheckStock = "SELECT S_QTY FROM STOCK
        WHERE S_W_ID = ? AND S_I_ID = ?";
    Query InsertOrder = "INSERT INTO ORDERS VALUES (?,?)";
    Query InsertOrdLine = "INSERT INTO ORDER_LINE VALUES (?,?,?,?)";
    Query UpdateStock = "UPDATE STOCK SET S_QTY = S_QTY - ?
        WHERE S_W_ID = ? AND S_I_ID = ?";

    int run(int w_id, int i_ids[], int i_w_ids[], int i_qtys[])
    {
        queueSQL(GetWarehouse, w_id);
        for (int i = 0; i < i_ids.length; i++)
            queueSQL(CheckStock, i_w_ids[i], i_ids[i]);
        Result r[] = executeBatch();
        int o_id = r[0].get("W_NEXT_O_ID") + 1;
        queueSQL(InsertOrder, w_id, o_id);
        for (int i = 0; i < r.length; i++)
            if (r[i+1].get("S_QTY") < i_qtys[i]) abort();
        queueSQL(InsertOrdLine, w_id, o_id, i_ids[i], i_qtys[i]);
        queueSQL(UpdateStock, i_qtys[i], i_w_ids[i], i_ids[i]);
    return (executeBatch() != null);
    }
Why this Matters

Assume Distributed  Assume Single-Partition  Magic Mode

Throughput (txn/s)

Number of Partitions
Pro Tip: Canadians do not like unnecessary surgeries.
On Predictive Modeling for Optimizing Transaction Execution in Parallel OLTP Systems

in PVLDB, vol 5. issue 2, October 2011
Main Idea:
Use models to predict before execution.
class NewOrder extends.StoredProcedure {
    Query GetWarehouse = "SELECT * FROM WAREHOUSE WHERE W_ID = ?";
    Query CheckStock = "SELECT S_QTY FROM STOCK WHERE S_W_ID = ? AND S_I_ID = ?";
    Query InsertOrder = "INSERT INTO ORDER_LINES VALUES (?, ?, ?, ?)";
    Query UpdateStock = "UPDATE STOCK SET S_QTY = ? WHERE S_W_ID = ? AND S_I_ID = ?";

    int run(int w_id, int i_ids[], int i_w_ids[], int i_qtys[]) {
        queueSQL(GetWarehouse, w_id);
        queueSQL(CheckStock, i_w_ids[i], i_ids[i]);
        Result r[] = executeBatch();
        int c_id = r[0].get("W_NEXT_O_ID") + 1;
        queueSQL(InsertOrder, w_id, c_id);
        for (int i = 0; i < i_ids.length; i++) {
            if (r[i].get("S_QTY") < i_qtys[i]) abort();
            queueSQL(UpdateStock, w_id, c_id, i_w_ids[i], i_qtys[i]);
        }
        return executeBatch(!= null);
    }
}
Step #1:
Estimate the path that a transaction will take
Step #2: Determine which optimizations to enable.
Current State:

```
begin
```

Input Parameters:
```
w_id=0
i_w_id=[0,1]  i_ids=[1001,1002]
```

GetWarehouse:
```
SELECT * FROM WAREHOUSE
WHERE W_ID = ?
```
begin

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

CheckStock
Counter: 0
Partitions: { 0 }
Previous: { 0 }

InsertOrder
Counter: 0
Partitions: { 0 }
Previous: { 0 }

UpdateStock
Counter: 0
Partitions: { 0 }
Previous: { 0 }

InsertOrdLine
Counter: 0
Partitions: { 0 }
Previous: { 0 }

commit

Input Parameters:

w_id=0
i_w_id=[0,1] i_ids=[1001,1002]

Transaction Estimate:

| Confidence Coefficient:       | 0.56 |
| Best Partition:               | 0    |
| Use Undo Logging:             | Yes  |
| Partitions Read:              | { 0  } |
| Partitions Written:           | { 0  } |
| Partitions Done:              | {1, 2, 3} |
Limitations:

1) Long/wide models.
2) Keeping models in synch.
3) Incorrect predictions.
SELECT S_QTY FROM STOCK WHERE S_W_ID = ? AND S_I_ID = ?;

Current State:
w_id=0 i_w_id=[0,1] i_ids=[1001,1002]

Input Parameters:
w_id=0 i_w_id=[0,1] i_ids=[1001,1002]

CheckStock:
SELECT S_QTY FROM STOCK WHERE S_W_ID = ? AND S_I_ID = ?;

InsertOrder:
INSERT INTO ORDERS (o_id, o_w_id) VALUES (?, ?);
THE GIFTS OF MISS CLEO

SUNDAY,
SEPTEMBER 16
9PM ET
ON IN DEMAND
PAY-PER-VIEW

Special Guest: Evan
GET A FREE 7 MINUTE TAROT READING
Refinement: Partition models based on input properties.
begin

GetWarehouse
   Counter: 0
   Partitions: { 0 }
   Previous: \emptyset

\begin{align*}
\text{CheckStock} & \quad \text{Counter: 0} \\
\text{CheckStock} & \quad \text{Counter: 0} \\
\text{CheckStock} & \quad \text{Counter: 1} \\
\text{CheckStock} & \quad \text{Counter: 1} \\
\text{InsertOrder} & \quad \text{Counter: 0}
\end{align*}

\begin{align*}
w_{id} = 0 \\
i_{w_id} = [0, 1] \quad i_{ids} = [1001, 1002]
\end{align*}

\textbf{CheckStock:}
\begin{align*}
\text{SELECT} & \quad S_{QTY} \\
\text{FROM} & \quad \text{STOCK} \\
\text{WHERE} & \quad S_{W_ID} = ? \\
\text{AND} & \quad S_{I_ID} = ?
\end{align*}
Houdini:

1) Execute txn at the best partition.
2) Only lock the partitions needed.
3) Disable undo logging if not needed.
4) Speculatively commit transactions.
Houdini:

5) Estimate initial path.
6) Update as transaction executes.
7) Recompute if workload changes.
8) Partition for better accuracy.
Experimental Evaluation
Model Accuracy

- TATP: 94.9%
- TPC-C: 95.0%
- AuctionMark: 90.2%
Estimation Overhead
Throughput

<table>
<thead>
<tr>
<th>(txn/s)</th>
<th>Assume Single-Partition</th>
<th>Global Model</th>
<th>Partitioned Models</th>
</tr>
</thead>
</table>

| Number of Partitions | TATP +57% | TPC-C +126% | AuctionMark +117% |

Graph showing throughput improvements for TATP, TPC-C, and AuctionMark with different numbers of partitions.
Conclusion:
Small overhead cost improves throughput.
Future work

class NewOrder extends StoredProcedure {
    Query GetWarehouse = "SELECT * FROM WAREHOUSE WHERE W ID = ?";
    Query CheckStock = "SELECT S QTY FROM STOCK
                        WHERE S W ID = ? AND S I ID = ?";
    Query InsertOrder = "INSERT INTO ORDERS VALUES (?,?)";
    Query InsertOrdLine = "INSERT INTO ORDER LINE VALUES (?,?,?,?)";
    Query UpdateStock = "UPDATE STOCK SET S QTY = S QTY - ?
                        WHERE S W ID = ? AND S I ID = ?";

    int run(int w_id, int i_ids[], int i_w_ids[], int i_qtys[]) {
        queueSQL(GetWarehouse, w_id);
        for (int i = 0; i < i_ids.length; i++)
            queueSQL(CheckStock, i_w_ids[i], i_ids[i]);
        Result r[] = executeBatch();
        int o_id = r[0].get("W_NEXT_O_ID") + 1;
        queueSQL(InsertOrder, w_id, o_id);
        for (int i = 0; i < r.length; i++) {
            if (r[i+1].get("S QTY") < i qtys[i]) abort();
            queueSQL(InsertOrdLine, w_id, o_id, i_ids[i], i_qtys[i]);
            queueSQL(UpdateStock, i_qtys[i], i_w_ids[i], i_ids[i]);
        }
        return (executeBatch() != null);
    }
}