Wander Join: Online Aggregation via Random Walks
Feifei Li (U. Utah), Bin Wu (HKUST), Ke Yi (HKUST), Zhuoyue Zhao (SJTU)

Abstract

Joins are expensive, and online aggregation over joins was proposed to mitigate the cost, which offers users a nice and flexible tradeoff between query efficiency and accuracy in a continuous, online fashion. We introduce a new approach, wander join, to the online aggregation problem by performing random walks over the underlying join graph.

Online Aggregation

```
SELECT ONLINE(SUM(extend_price * (1 - 1_discount)) AS online_cost FROM orders, lineitem WHERE order_status = 'active' AND order_key = order_key GROUP BY order_key, lineitem_key ORDER BY online_cost DESC LIMIT 100;
```

Wander Join

```
WITH WJ AS
(
    SELECT order_key, customer_key, order_status, order_timestamp, order_total_price
    FROM orders
),

RJ AS
(
    SELECT R1.order_key, R1.customer_key, R1.order_status, R1.order_timestamp, R1.order_total_price
    FROM WJ AS W1
    JOIN WJ AS W2 ON W1.customer_key = W2.customer_key
),

Q3 AS
(
    SELECT W1.order_key, W1.customer_key, W1.order_status, W1.order_timestamp, W1.order_total_price
    FROM WJ AS W1
    INNER JOIN RJ AS R1 ON W1.customer_key = R1.customer_key
    WHERE R1.order_status = 'active'
),

Q7 AS
(
    SELECT W1.order_key, W1.customer_key, W1.order_status, W1.order_timestamp, W1.order_total_price
    FROM WJ AS W1
    INNER JOIN RJ AS R1 ON W1.customer_key = R1.customer_key
    WHERE R1.order_status = 'active'
),

Q10 AS
(
    SELECT W1.order_key, W1.customer_key, W1.order_status, W1.order_timestamp, W1.order_total_price
    FROM WJ AS W1
    INNER JOIN RJ AS R1 ON W1.customer_key = R1.customer_key
    WHERE R1.order_status = 'active'
)

```

Comparison

Wander join

- Independent but non-uniform
- Sampling methodology
- Uniform but non-independent
- Needed
- Index
- Not needed
- Easy
- Confidence interval computation
- Complicated
- $O(\text{n})$ Time
- $O(k)$ Time
- $~3$ seconds
- $~100$ seconds
- $~50$ seconds
- $~100$ seconds
- Logarithmic
- Scalability
- Linear
- Parser
- Component affected
- Almost everything
- Informix
- (CONTROL project)
- DBO

Standalone implementation.

```
<table>
<thead>
<tr>
<th>Nation</th>
<th>CID</th>
<th>BuyerID</th>
<th>OrderID</th>
<th>OrderID</th>
<th>ItemID</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>301</td>
<td>$300</td>
</tr>
<tr>
<td>US</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>304</td>
<td>$100</td>
</tr>
<tr>
<td>China</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>201</td>
<td>$300</td>
</tr>
<tr>
<td>UK</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>306</td>
<td>$500</td>
</tr>
<tr>
<td>China</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>$230</td>
</tr>
<tr>
<td>US</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>101</td>
<td>$800</td>
</tr>
<tr>
<td>China</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>201</td>
<td>$300</td>
</tr>
<tr>
<td>UK</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>101</td>
<td>$200</td>
</tr>
<tr>
<td>Japan</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>301</td>
<td>$100</td>
</tr>
<tr>
<td>UK</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>301</td>
<td>$100</td>
</tr>
</tbody>
</table>
```

Unbiased estimator:

$$\frac{\$500 \cdot \text{sampling prob.}}{1/3 \cdot 1/4 \cdot 1/3} = \$500$$

Walk Plan Optimizer

```
R_1 \rightarrow R_2 \rightarrow R_3
```

All possible walk plans.

Structure of the join data graph has a significant impact on the performance of different walk plans.

System implementation

```
<table>
<thead>
<tr>
<th>CI: half width of the confidence interval (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System X: full join on System X (seconds)</td>
</tr>
<tr>
<td>CI: half width of the confidence interval (%)</td>
</tr>
<tr>
<td>10: 32.24 (0.76) 0.26 107.27 15.9 7.8</td>
</tr>
<tr>
<td>20: 74.29 (0.78) 0.43 249.94 11.1 4.3</td>
</tr>
<tr>
<td>30: 65.17 (0.84) 0.40 428.39 9.6 4.5</td>
</tr>
<tr>
<td>40: 90.23 (0.76) 0.26 707.04 8.1 4.7</td>
</tr>
</tbody>
</table>

Accuracy achieved in 1/10 of System X’s full running time.
```

```
<table>
<thead>
<tr>
<th>size(GB)</th>
<th>System X (X)</th>
<th>PG+WJ</th>
<th>System X (X)</th>
<th>PG+WJ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\text{CI \ AE})</td>
<td>(\text{AE})</td>
<td>(\text{CI \ AE})</td>
<td>(\text{AE})</td>
</tr>
<tr>
<td>10</td>
<td>32.24 (0.76) 0.26 107.27 15.9 7.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>74.29 (0.78) 0.43 249.94 11.1 4.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>65.17 (0.84) 0.40 428.39 9.6 4.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>90.23 (0.76) 0.26 707.04 8.1 4.7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accuracy achieved in 1/10 of System X’s running time for computing the full join.

1. System X: full join on System X (seconds).
2. CI: half width of the confidence interval (%).
3. AE: actual error (%).
4. PG+WJ: Our version of PostgreSQL with Wander Join implemented inside the PostgreSQL engine.