Locality-Sensitive Operators for Parallel Main-Memory Database Clusters

Wolf Rödiger, Tobias Mühlbauer, Philipp Unterbrunner*, Angelika Reiser, Alfons Kemper, Thomas Neumann

Technische Universität München, *Snowflake Computing, Inc.
Scale Out

- **HyPer**: High-performance in-memory transaction and query processing system
- **Scale out** to process very large inputs
- Aim at **clusters** with large main memory capacity
- A server with 20 cores and 256 GB RAM costs \(\sim\)$7,500
Running Example (1)

- Focus on **analytical** query processing in this talk
- TPC-H query 12 used as **running example**
- Runtime dominated by **join** orders ⊗ lineitem
Running Example (2)

- Relations are **equally** distributed across nodes
- We make **no** assumptions on the data distribution
- Thus, tuples may join with tuples on **remote** nodes
- **Communication** over the network required
CPU speed has grown much faster than network bandwidth
Scale Out: Network is the Bottleneck

- **Single node**: Performance is bound algorithmically
- **Cluster**: Network is bottleneck for query processing
- We propose a novel join algorithm called **Neo-Join**
- **Goal**: Increase local processing to close the performance gap
Neo-Join: Network-optimized Join

1. **Open Shop Scheduling**
   Efficient network communication

2. **Optimal Partition Assignment**
   Increase local processing

3. **Selective Broadcast**
   Handle value skew
Open Shop Scheduling

Efficient network communication
Standard Network Model

- **Star topology**
  Nodes are connected to a central switch

- **Fully switched**
  All links can be used simultaneously

- **Fully duplex**
  Nodes can both send and receive at full speed
Bandwidth Sharing

- Simultaneous use of a single link creates a bottleneck
- Reduces bandwidth by at least a factor of 2
Naïve Schedule

- Node 2 and 3 send to node 1 at the same time
- Bandwidth sharing increases network duration significantly
Avoiding bandwidth sharing translates to **open shop scheduling**: 

- A **job** consists of one **task** per **processor**
- A processor can perform at most **one** task at a time
- At most **one** task of a job can be processed at a time
Avoiding bandwidth sharing translates to open shop scheduling:

- A sender has one transfer per receiver
- A receiver should receive at most one transfer at a time
- A sender should send at most one transfer at a time
Compute optimal schedule:

- **Edge weights** represent total transfer duration
- Scheduler repeatedly finds **perfect matchings**
- Each matching specifies one communication **phase**
- Transfers in a phase will **never** share bandwidth
Optimal Schedule

- Open shop schedule achieves minimal network duration
- Schedule duration determined by maximum straggler

Wolf Rödiger
Locality-Sensitive Operators for Parallel Main-Memory Database Clusters
Optimal Partition Assignment

Minimize network duration for distributed joins
Distributed Join

- Tuples may join with tuples on **remote nodes**
- Repartition and redistribute **both relations** for local join
- Tuples will join only with the **corresponding partition**
- Using hash, range, radix, or other **partitioning** scheme
- **In any case:** Decide how to **assign** partitions to nodes
Running Example: Hash Partitioning

orders
key priority
1 1-URGENT
2 2-HIGH
3 1-URGENT
4 5-LOW
5 3-MEDIUM
6 1-URGENT
7 2-HIGH
8 1-URGENT
9 1-URGENT
10 2-HIGH
11 3-MEDIUM
12 5-LOW
13 1-URGENT
14 3-MEDIUM
15 1-URGENT
16 3-MEDIUM
17 2-HIGH
18 3-MEDIUM
19 5-LOW
20 1-URGENT
21 2-HIGH

lineitem
key shipmode
1 MAIL
1 MAIL
1 MAIL
2 SHIP
2 MAIL
6 SHIP
6 SHIP
6 SHIP
6 MAIL
10 SHIP
11 MAIL
11 MAIL
13 MAIL
13 MAIL
13 MAIL
13 SHIP
17 MAIL
18 MAIL
18 MAIL
19 SHIP
20 SHIP

x+2 mod 3

node 1
node 2
node 3
Assign Partitions to Nodes (1)

**Option 1:** Minimize network traffic

- Assign partition to node that owns its **largest part**
- Only the **small fragments** of a partition sent over the network
- Schedule with minimal network traffic may have **high duration**

![Diagram showing hash partitioning (x mod 3) and open shop schedule with traffic and time values]
Assign Partitions to Nodes (2)

Option 2: Minimize response time:

- Query response time is time from request to result
- Query response time dominated by network duration
- To minimize network duration, minimize maximum straggler

hash partitioning (x mod 3)

<table>
<thead>
<tr>
<th>n1</th>
<th>n2</th>
<th>n3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

open shop schedule

<table>
<thead>
<tr>
<th>n1</th>
<th>n2</th>
<th>n3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

traffic: 28  time: 10
Minimize Maximum Straggler

- Formalized as mixed-integer linear program
- Shown to be \textbf{NP-hard} (see paper for proof sketch)
- In practice \textbf{fast enough} using CPLEX or Gurobi \newline \hspace{0.3cm} (< 0.5% overhead for 32 nodes, 200 M tuples each)
- Partition assignment can optimize \textbf{any partitioning}

\[\begin{align*}
\text{minimize } w, \text{ subject to } \\
v & \geq \sum_{j=0}^{p-1} h_{ij} (1 - x_{ij}) \quad 0 \leq i < n \\
w & \geq \sum_{j=0}^{p-1} \left( x_{ij} \sum_{k=0, i \neq k}^{n-1} h_{kj} \right) \quad 0 \leq i < n \\
1 & = \sum_{i=0}^{n-1} x_{ij} \quad 0 \leq j < p
\end{align*}\]
Running Example: Locality

<table>
<thead>
<tr>
<th>node 1</th>
<th>orders key</th>
<th>orders priority</th>
<th>lineitem key</th>
<th>lineitem shipmode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-URGENT</td>
<td>2</td>
<td>1</td>
<td>MAIL</td>
</tr>
<tr>
<td>2</td>
<td>2-HIGH</td>
<td>3</td>
<td>1</td>
<td>MAIL</td>
</tr>
<tr>
<td>3</td>
<td>1-URGENT</td>
<td>4</td>
<td>1</td>
<td>MAIL</td>
</tr>
<tr>
<td>4</td>
<td>5-LOW</td>
<td>5</td>
<td>1</td>
<td>MAIL</td>
</tr>
<tr>
<td>5</td>
<td>3-MEDIUM</td>
<td>6</td>
<td>1</td>
<td>MAIL</td>
</tr>
<tr>
<td>6</td>
<td>1-URGENT</td>
<td>7</td>
<td>2</td>
<td>SHIP</td>
</tr>
<tr>
<td>7</td>
<td>2-HIGH</td>
<td>8</td>
<td>1</td>
<td>MAIL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node 2</th>
<th>orders key</th>
<th>orders priority</th>
<th>lineitem key</th>
<th>lineitem shipmode</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1-URGENT</td>
<td>10</td>
<td>11</td>
<td>MAIL</td>
</tr>
<tr>
<td>10</td>
<td>2-HIGH</td>
<td>11</td>
<td>11</td>
<td>MAIL</td>
</tr>
<tr>
<td>11</td>
<td>3-MEDIUM</td>
<td>12</td>
<td>11</td>
<td>MAIL</td>
</tr>
<tr>
<td>12</td>
<td>5-LOW</td>
<td>13</td>
<td>11</td>
<td>MAIL</td>
</tr>
<tr>
<td>13</td>
<td>1-URGENT</td>
<td>14</td>
<td>11</td>
<td>MAIL</td>
</tr>
<tr>
<td>14</td>
<td>3-MEDIUM</td>
<td>15</td>
<td>11</td>
<td>MAIL</td>
</tr>
<tr>
<td>15</td>
<td>1-URGENT</td>
<td>16</td>
<td>15</td>
<td>MAIL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node 3</th>
<th>orders key</th>
<th>orders priority</th>
<th>lineitem key</th>
<th>lineitem shipmode</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>3-MEDIUM</td>
<td>17</td>
<td>13</td>
<td>MAIL</td>
</tr>
<tr>
<td>17</td>
<td>2-HIGH</td>
<td>18</td>
<td>13</td>
<td>SHIP</td>
</tr>
<tr>
<td>18</td>
<td>3-MEDIUM</td>
<td>19</td>
<td>13</td>
<td>MAIL</td>
</tr>
<tr>
<td>19</td>
<td>5-LOW</td>
<td>20</td>
<td>18</td>
<td>MAIL</td>
</tr>
<tr>
<td>20</td>
<td>1-URGENT</td>
<td>21</td>
<td>18</td>
<td>MAIL</td>
</tr>
<tr>
<td>21</td>
<td>2-HIGH</td>
<td></td>
<td>18</td>
<td>MAIL</td>
</tr>
</tbody>
</table>

The diagram shows the distribution of keys across nodes and the radix operations for each node.
Locality

- Running example exhibits time-of-creation clustering
- **Radix repartitioning** on most significant bits retains locality
- Partition assignment can exploit locality
- Significantly reduces query response time
Selective Broadcast

Handle value skew
Running Example: Skew

<table>
<thead>
<tr>
<th>orders</th>
<th>lineitem</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>priority</td>
</tr>
<tr>
<td>1</td>
<td>1-URGENT</td>
</tr>
<tr>
<td>2</td>
<td>2-HIGH</td>
</tr>
<tr>
<td>3</td>
<td>1-URGENT</td>
</tr>
<tr>
<td>4</td>
<td>1-URGENT</td>
</tr>
<tr>
<td>node 1</td>
<td>node 3</td>
</tr>
<tr>
<td>5</td>
<td>1-URGENT</td>
</tr>
<tr>
<td>6</td>
<td>2-HIGH</td>
</tr>
<tr>
<td>7</td>
<td>3-MEDIUM</td>
</tr>
<tr>
<td>node 2</td>
<td>node 1</td>
</tr>
<tr>
<td>8</td>
<td>3-MEDIUM</td>
</tr>
<tr>
<td>9</td>
<td>2-HIGH</td>
</tr>
<tr>
<td>node 3</td>
<td>node 1</td>
</tr>
</tbody>
</table>

x+2 mod 3

O₁ L₁ O₂ L₂ O₃ L₃
Skew

- **Skewed** partition $P_2$ has to be assigned, e.g., to node 3
- Node 3 will receive **much more** than its fair share
- May balance skewed partitions by creating **more partitions**
- **However:** More expensive and **high skew** is still a problem
Broadcast

- **Alternative** to data repartitioning
- **Replicate** the smaller relation between all nodes
- Larger relation **remains fragmented** across nodes
Selective Broadcast

- Decide **per partition** whether to assign or broadcast
- **Broadcast** orders for $P_2$, let line items remain fragmented
- **Assign** the other partitions taking locality into account
- Improves performance for high skew and many duplicates

### hash partitioning (mod 3)

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$L_1$</th>
<th>$O_2$</th>
<th>$L_2$</th>
<th>$O_3$</th>
<th>$L_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$n_2$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$n_3$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### open shop schedule

<table>
<thead>
<tr>
<th></th>
<th>$n_1$</th>
<th>$n_2$</th>
<th>$n_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_1$</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$L_1$</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$O_2$</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$L_2$</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$O_3$</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$L_3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**traffic**: 14  
**time**: 6
Role Reversal

- Selective broadcast allows for role reversal
- Broadcast different partitions by different relations

Example:

- **Large suppliers** produce a large variety of parts
- **Important parts** available from many suppliers
Evaluation
Experimental Setup

- Cluster of 4 nodes
- Core i7, 4 cores, 3.4 GHz, 32 GB RAM
- Gigabit Ethernet
- Tuples consist of 64 bit key, 64 bit payload
Locality

- Vary **locality** from 0% (uniform distribution) to 100% (range partitioning)
- Neo-Join improves **join performance** from 29 M to 156 M tuples/s (> 500%)
- 3 nodes, 600 M tuples

![Graph showing join performance with different localities and DBMS]
Skew

- **Zipfian distribution** models realistic data skew
- Using **more partitions** alleviates the problem
- Selective broadcast actually **improves** performance for skewed inputs
- 4 nodes, 400 M tuples

<table>
<thead>
<tr>
<th>Zipf factor s</th>
<th>partitions</th>
<th>0.00</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>27s</td>
<td>24s</td>
<td>23s</td>
<td>29s</td>
<td>44s</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>23s</td>
<td>23s</td>
<td>23s</td>
<td>23s</td>
<td>33s</td>
<td></td>
</tr>
<tr>
<td>16 (SB)</td>
<td>24s</td>
<td>24s</td>
<td>23s</td>
<td>20s</td>
<td>10s</td>
<td></td>
</tr>
</tbody>
</table>
TPC-H Results (scale factor 100)

- Results for three selected TPC-H queries
- **Broadcast** outperforms **hash** for large relation size differences
- Neo-Join always performs better due to **selective broadcast** and **locality**
- 4 nodes, scale factor 100
Summary

Motivation:

- **Scale out** to handle very large inputs
- **Network** is the bottleneck
- Thus, **reduce** network duration

Contributions:

- Maximize bandwidth usage with **Open Shop Scheduling**
- Exploit locality with **Optimal Partition Assignment**
- Handle skewed inputs with **Selective Broadcast**