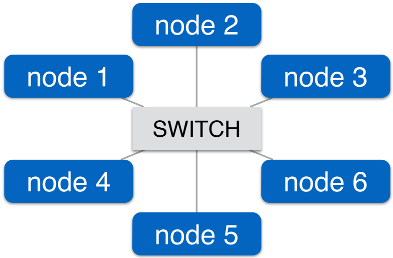
Locality-Sensitive Operators for Parallel Main-Memory Database Clusters

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

Technische Universität München, *Snowflake Computing, Inc.

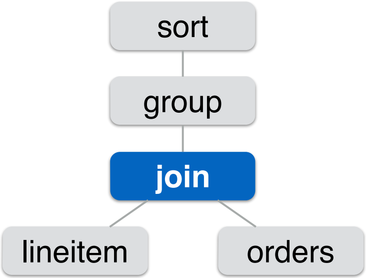
Scale Out

- **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 ~$7,500



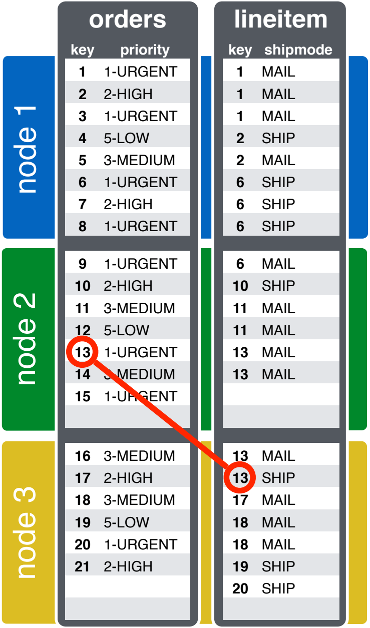
Running Example (1)

- Focus on **analytical** query processing in this talk
- TPC-H query 12 used as **running example**
- Runtime dominated by **join** orders $\bowtie$ lineitem

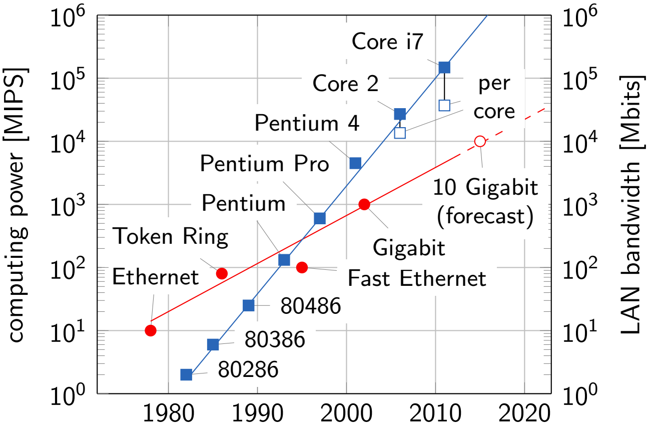


Running Example (2)

- 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



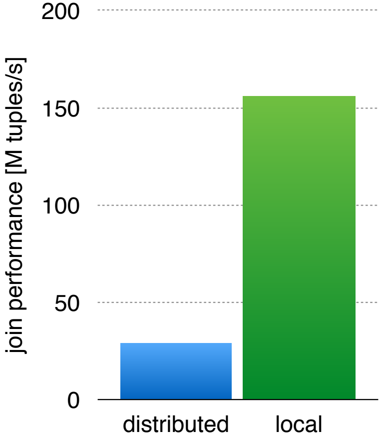
CPU vs. Network



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

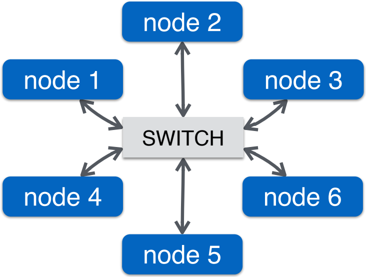
3. **Selective Broadcast**
   Handle value skew

Open Shop Scheduling

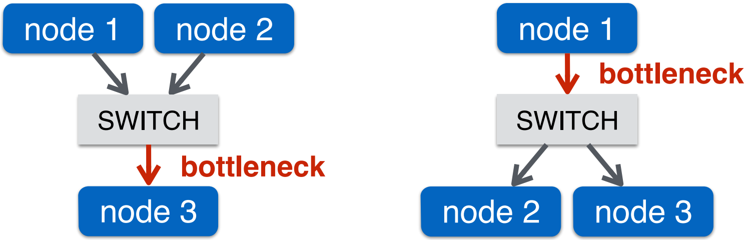
Efficient network communication

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

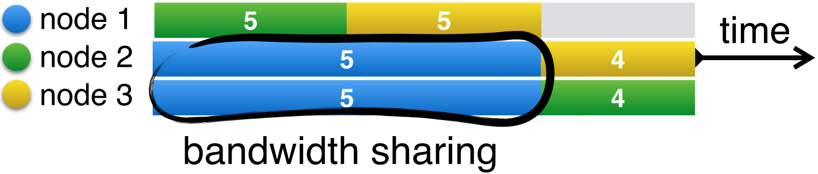


Bandwidth Sharing



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

Naïve Schedule

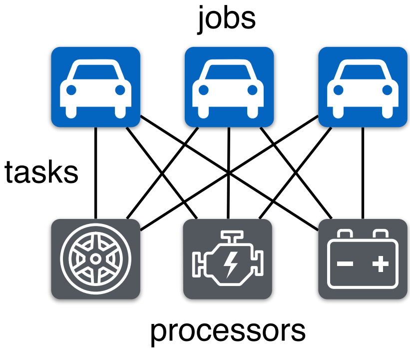


- Node 2 and 3 send to node 1 **at the same time**
- Bandwidth sharing increases **network duration** significantly

Open Shop Scheduling (1)

Avoiding bandwidth sharing translates to **open shop scheduling:**

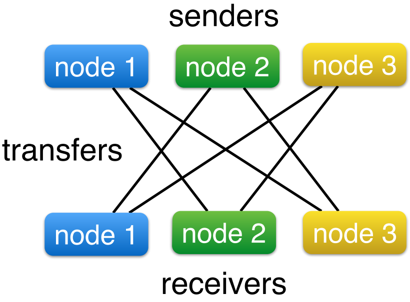
- A **job** consists of one **task** per **processor**
- A processor can perform at most **one** task at a time
- At most **one** task of a job can be processed at a time



Open Shop Scheduling (2)

Avoiding bandwidth sharing translates to **open shop scheduling:**

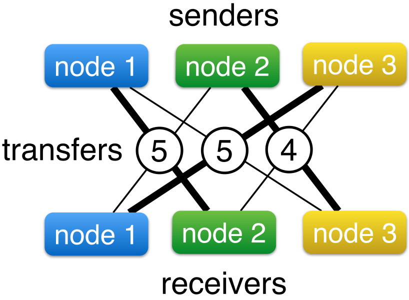
- A **sender** has one **transfer** per **receiver**
- A receiver should receive at most **one** transfer at a time
- A sender should send at most **one** transfer at a time



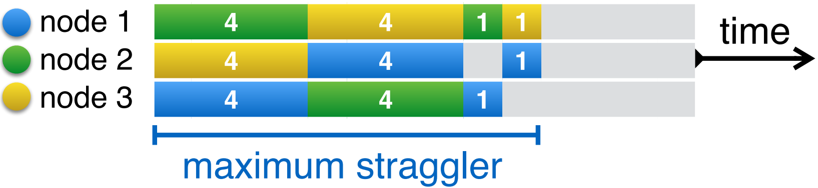
Open Shop Scheduling (3)

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



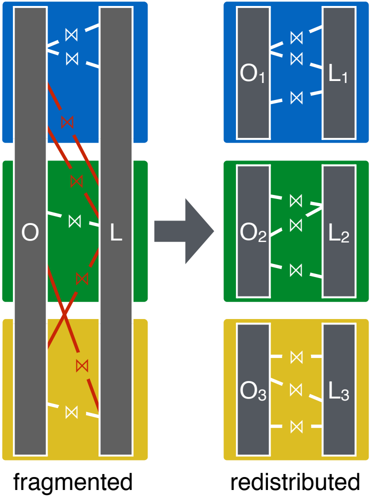
- Open shop schedule achieves minimal **network duration**
- Schedule duration determined by **maximum straggler**

Optimal Partition Assignment

Minimize network duration for distributed joins

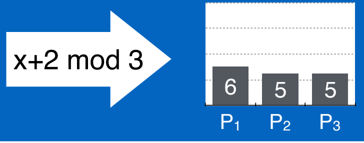
Distributed Join

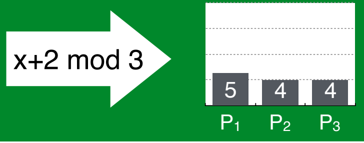
- Tuples may join with tuples on **remote nodes**
- Repartition and redistribute **both relations** for local join
- Tuples will join only with the **corresponding partition**
- Using hash, range, radix, or other **partitioning** scheme
- **In any case:** Decide how to **assign** partitions to nodes

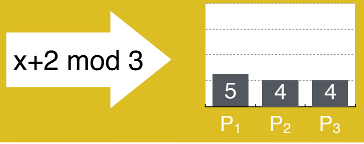


Running Example: Hash Partitioning

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



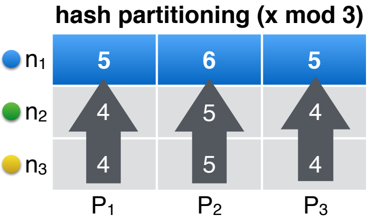


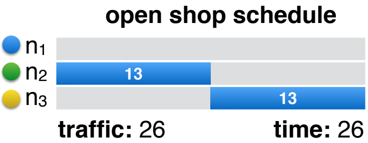


Assign Partitions to Nodes (1)

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**

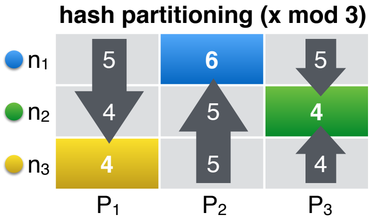


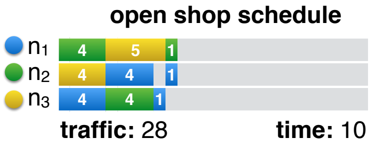


Assign Partitions to Nodes (2)

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**





Minimize Maximum Straggler

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

minimize w, subject to

$$w \geq \sum_{j=0}^{p-1} h_{ij}(1 - x_{ij}) \qquad 0 \leq i < n$$

$$w \geq \sum_{j=0}^{p-1} \left(x_{ij} \sum_{k=0, i \neq k}^{n-1} h_{kj} \right) \qquad 0 \leq i < n$$

$$1 = \sum_{i=0}^{n-1} x_{ij} \qquad 0 \leq j < p$$

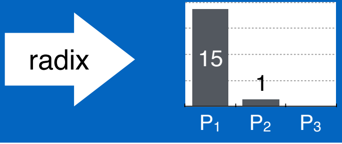
Running Example: Locality

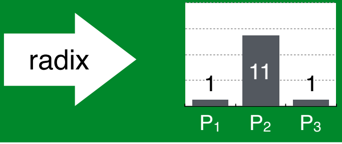
orders

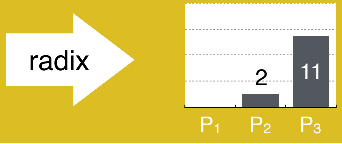
node	key	priority
node 1	**1**	1-URGENT
node 1	**2**	2-HIGH
node 1	**3**	1-URGENT
node 1	**4**	5-LOW
node 1	**5**	3-MEDIUM
node 1	**6**	1-URGENT
node 1	**7**	2-HIGH
node 1	**8**	1-URGENT
node 2	**9**	1-URGENT
node 2	**10**	2-HIGH
node 2	**11**	3-MEDIUM
node 2	**12**	5-LOW
node 2	**13**	1-URGENT
node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

lineitem

node	key	shipmode
node 1	**1**	MAIL
node 1	**1**	MAIL
node 1	**1**	MAIL
node 1	**2**	SHIP
node 1	**2**	MAIL
node 1	**6**	SHIP
node 1	**6**	SHIP
node 1	**6**	SHIP
node 2	**6**	MAIL
node 2	**10**	SHIP
node 2	**11**	MAIL
node 2	**11**	MAIL
node 2	**13**	MAIL
node 2	**13**	MAIL
node 3	**13**	MAIL
node 3	**13**	SHIP
node 3	**17**	MAIL
node 3	**18**	MAIL
node 3	**18**	MAIL
node 3	**19**	SHIP
node 3	**20**	SHIP

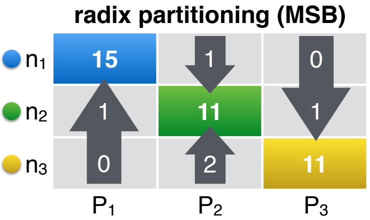






Locality

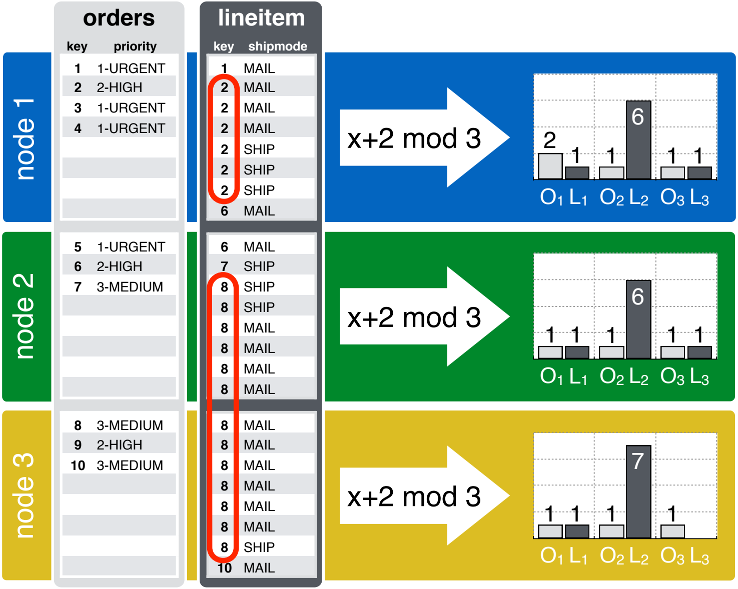
- Running example exhibits **time-of-creation** clustering
- **Radix repartitioning** on most significant bits retains locality
- Partition assignment can **exploit locality**
- Significantly reduces **query response time**



Selective Broadcast

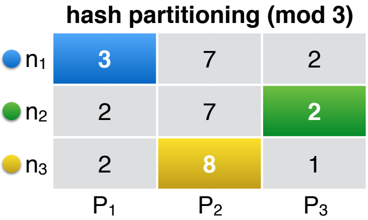
Handle value skew

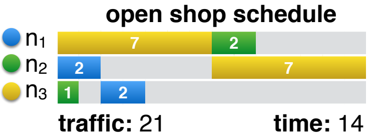
Running Example: Skew



Skew

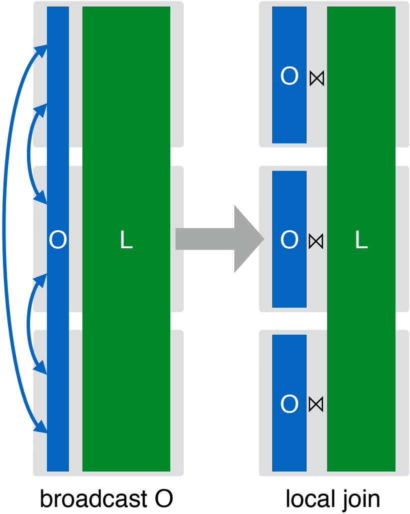
- **Skewed** partition P_2 has to be assigned, e.g., to node 3
- Node 3 will receive **much more** than its fair share
- May balance skewed partitions by creating **more partitions**
- **However:** More expensive and **high skew** is still a problem





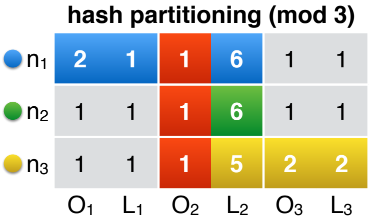
Broadcast

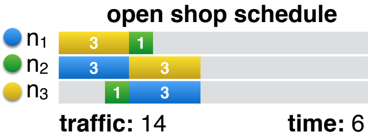
- **Alternative** to data repartitioning
- **Replicate** the smaller relation between all nodes
- Larger relation **remains fragmented** across nodes



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**



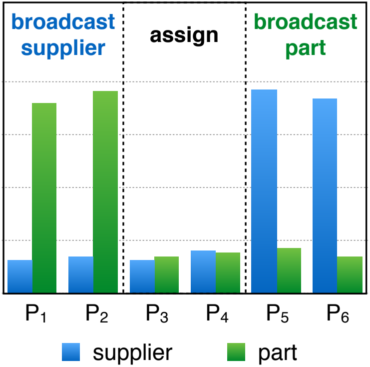


Role Reversal

- 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



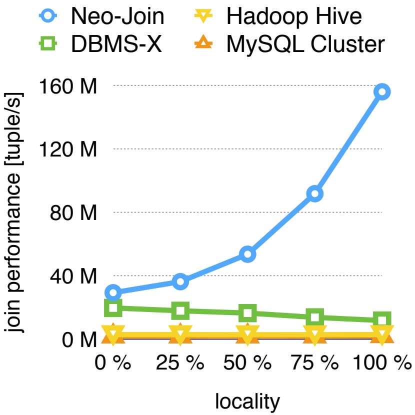
Evaluation

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



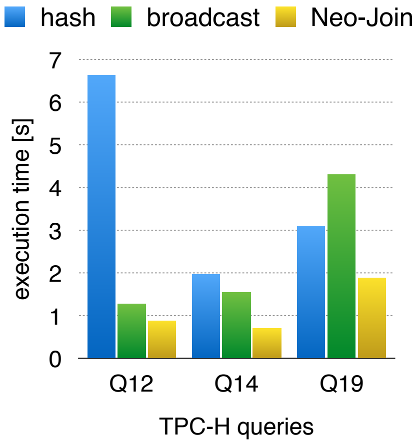
Skew

- **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

	Zipf factor s				
partitions	0.00	0.25	0.50	0.75	1.00
16	27 s	24 s	23 s	**29s**	**44s**
512	23 s	23 s	23 s	23 s	**33s**
16 (SB)	24 s	24 s	23 s	**20s**	**10s**

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**