HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

An RDMA Middleware for Multi-stage Asynchronous Shuffling in Analytical Processing

Rui C. Gonçalves[1], José Pereira[1], Ricardo Jiménez-Peris[2]
[1]INESC TEC & U. Minho
[2]Univ. Politécnica de Madrid & LeanXcale

Distributed Applications and Interoperable Systems
June 2016

INESCTEC
TECNOLOGIA E CIÊNCIA
LABORATÓRIO ASSOCIADO

Universidade do Minho

Context

- Big Data: Explosion of data being generated and stored every day
  - Holds valuable knowledge for organizations' operation
  - Opportunity to improve their efficiency
  - New challenges to store and process massive amounts of data

Context

- Emergence of new solutions for large scale data processing, as alternatives to traditional Relational Database Management Systems
  - NoSQL database systems
  - MapReduce programming model
  - Stream processing frameworks
- New solutions that offer SQL-like interfaces
  - Hive
  - Impala

Shuffling

- Shuffling is a key concept in multiple solutions
  - Provides distribution of data for parallel processing, possibly aggregating related items
- Different shuffling approaches
  - Push-based vs pull-based
  - Synchronous vs asynchronous
  - Hash code vs random
- Requires distributed coordination
- Critical for systems performance

Shuffling

- Shuffling is a key concept in multiple solutions
  - Provides distribution of data for parallel processing, possibly aggregating related items
- Different shuffling approaches
  - Push-based vs pull-based
  - Synchronous vs asynchronous
  - Hash code vs random
- Requires distributed coordination
- Cri

Leverage from RDMA protocols to improve shuffling

RDMA Protocols

- Low latency, high throughput communications
- Direct application access to network hardware
- Zero-copy communications
- Multiple RDMA technologies
  - Software implementations also available: e.g. Soft-iWARP and Soft-RoCE

RDMA Verbs

- Asynchronous API to use RDMA protocols
  - Requests are queued, and completion events may be generated when requests are completed
- Two types of communication semantics:
  - Memory semantics (one-sided read/write operations)
  - Channel semantics (two-sided send/receive operations)
- Predefined memory locations
  - Requires redesigning applications currently using socket-based communications

Shuffling for Analytical Processing

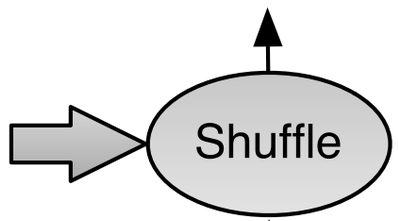
Shuffling for Analytical Processing

- Shuffling implementation to support analytical workloads on a Java distributed query engine (SQL)
- Parallelization of stateful operators requires shuffling to group related rows

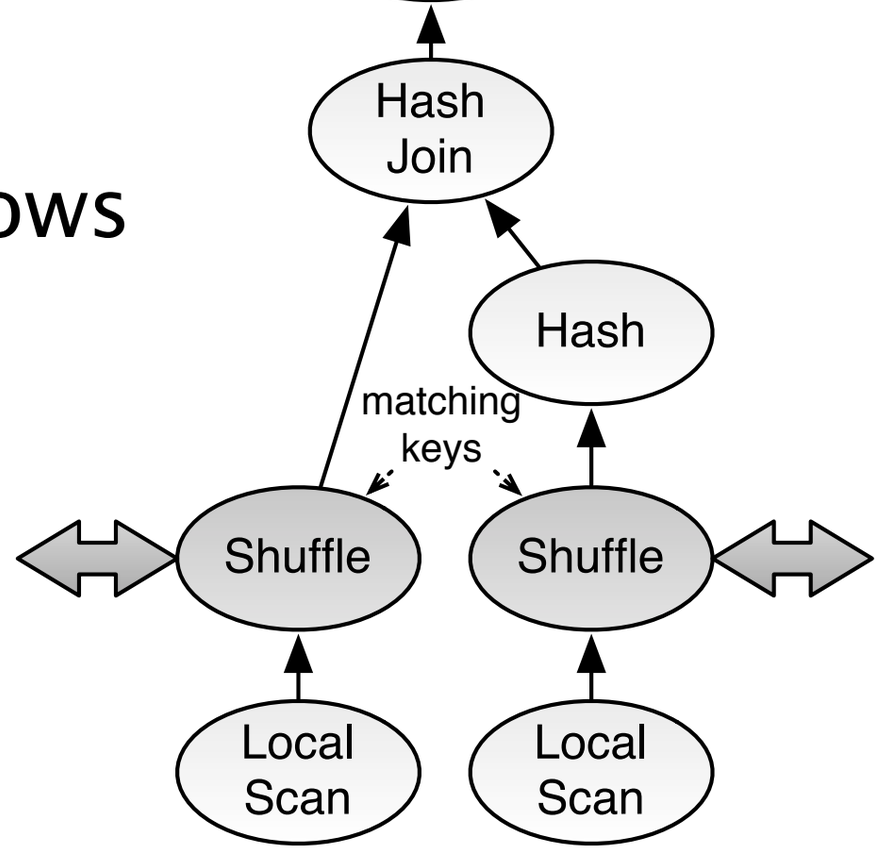
Shuffling for Analytical Processing

- Shuffling implementation to support analytical workloads on a Java distributed query engine (SQL)

```
select *
  from a inner join b
    on a.x = b.x
```



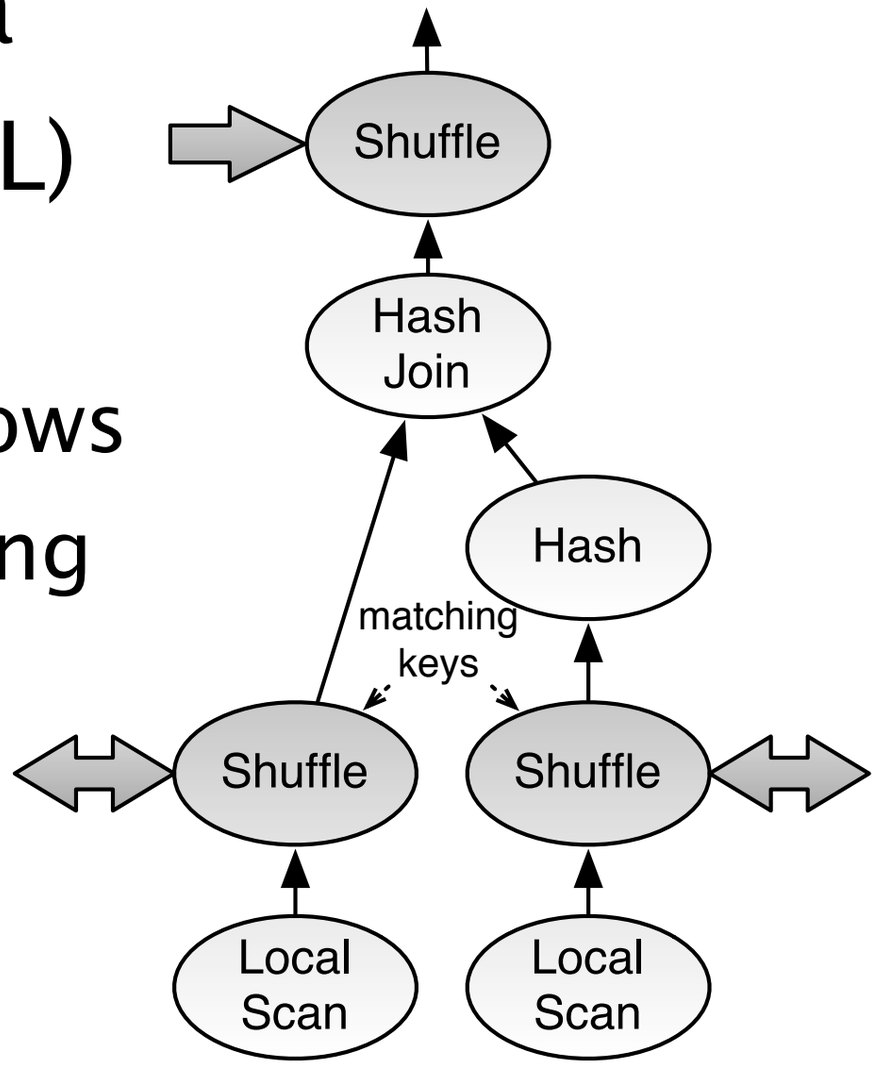
- Parallelization of stateful operators requires shuffling to group related rows



Shuffling for Analytical Processing

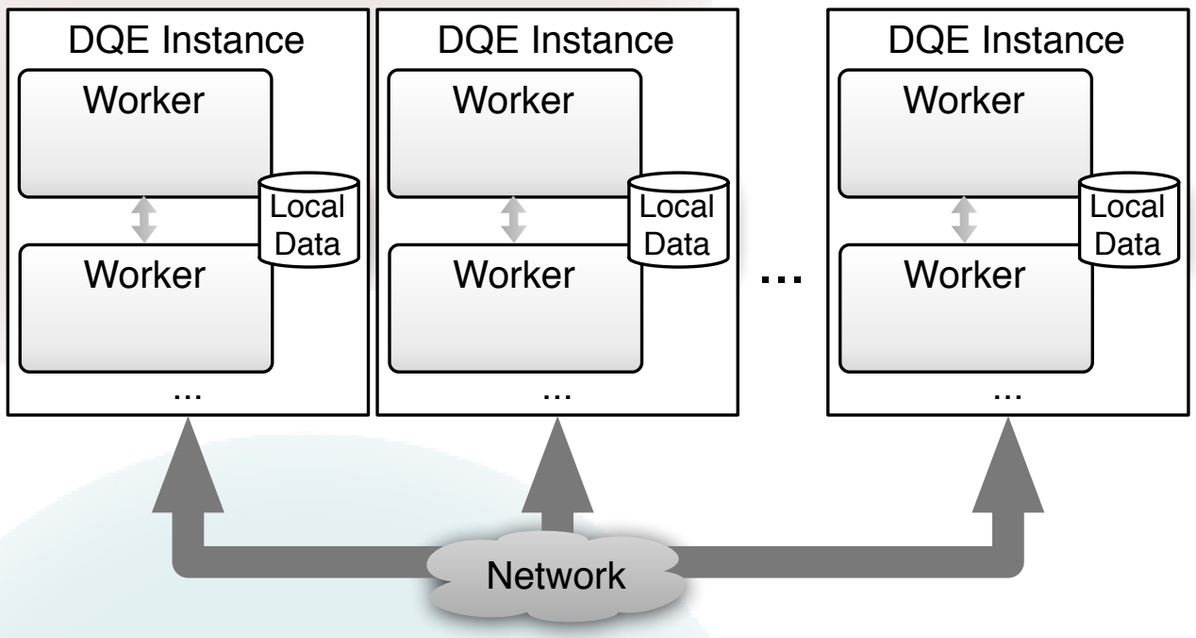
- Shuffling implementation to support analytical workloads on a Java distributed query engine (SQL)
- Parallelization of stateful operators requires shuffling to group related rows
- Multi-stage, to allow multiple shuffling steps
- Asynchronous, to allow tasks to execute in parallel at different steps

```
select *
  from a inner join b
    on a.x = b.x
```



DQE Analytical Processing Architecture

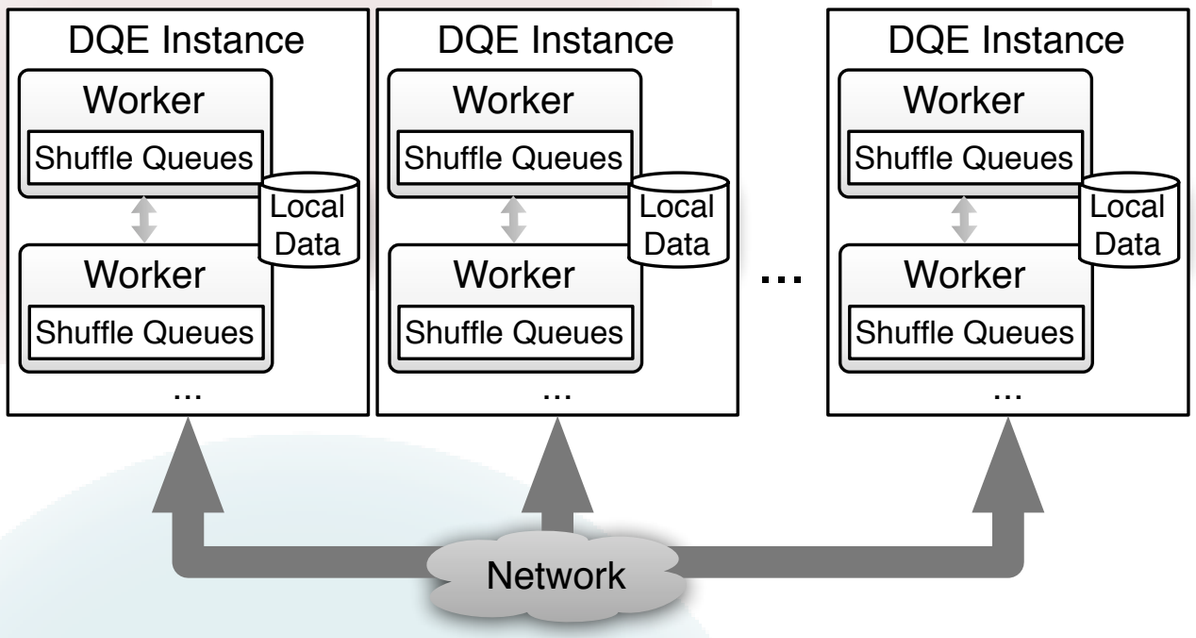
- Workers execute the operators of the query plan, which requires fetching rows from child operators
- When fetching rows from Shuffle operators:
  - It may return a row received from other worker
  - If there is no received row, it fetches a row from its child operator
    - The row fetched may be returned or shuffled



HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

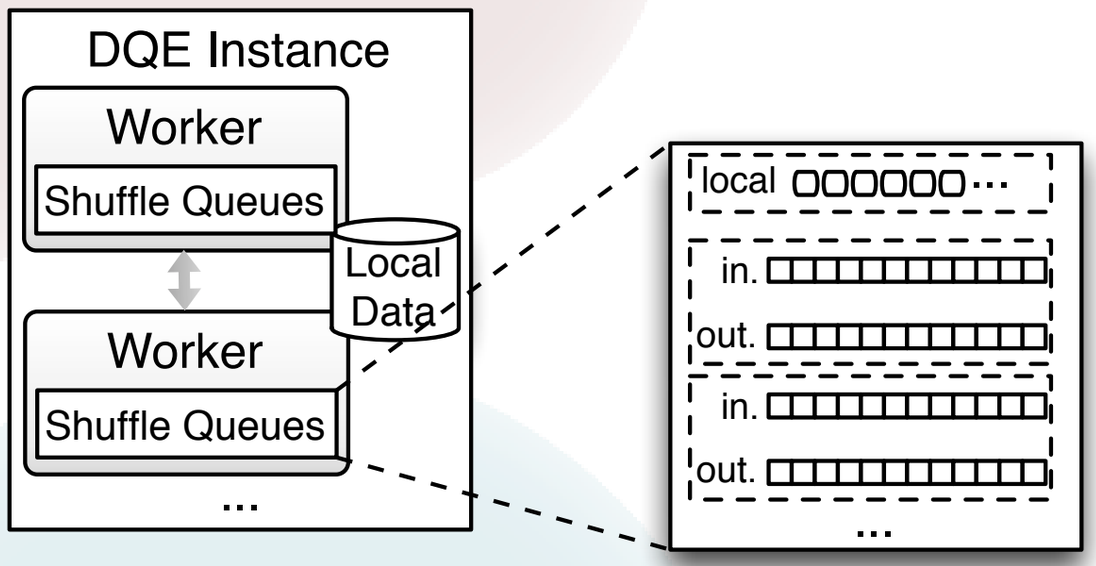
Asynchronous Shuffling

- Push-based asynchronous shuffling
  - Rows are sent as soon as they are processed by the Shuffle operator
  - Workers maintain **Shuffle Queues**, which receive remote rows and store them until they are consumed



Shuffle Queues

- Contain an incoming and an outgoing circular buffer per each remote thread
  - Outgoing buffers are used to serialize rows to send
  - Incoming buffers are used to receive rows and store it until they are consumed
  - Rows from local workers are put on a dynamic queue
- Communication middleware transfers data from an outgoing buffer to the matching incoming buffer



HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Communication Middleware

- Key functionalities required
  - Ability to send (and queue) rows to remote workers
  - Ability to retrieve queued rows
  - Ability to block a worker when there are no rows to process (and to wake it up when new rows become available)
  - Ability to block a worker when the local outgoing buffers are full (and to wake it up when space becomes available)

RDMA Middleware Overview

- RDMA Write operations to transfer data
- Send/Receive operations for notifications
- Dedicated network thread on each DQE instance
- Tracks requests' completion events
- Manages notifications

Workers

- When rows to shuffle become available:
  - Serialize rows and post write requests, unless:
    - Outgoing buffer is full (worker blocks)
    - Remote incoming buffer is full (worker proceeds with its operation)
- When polling rows from shuffle queues:
  - Check for rows received in local queue
  - If no row was available, poll incoming buffers
    - When data is removed from buffers, notifies the remote side, so that space released can be reused
  - Block if shuffle queues are empty and child operator has no more rows

Network Thread

- In charge of operations that follow the completion event of network operations
- After completion of write operations
  - Notifies the remote side that data was written
  - Releases space on outgoing buffers
    - Wakes up workers blocked for writing

Network Thread

- After receiving notifications
  - Maintains a queue of buffers with data available based on write notifications (to avoid the need of active polling)
    - Wakes up workers blocked for reading
  - Maintains the space available on remote buffers based on read notifications (to determine whether write requests can be performed)
    - Post write requests for data on buffers (if worker was unable to post a write request previously)

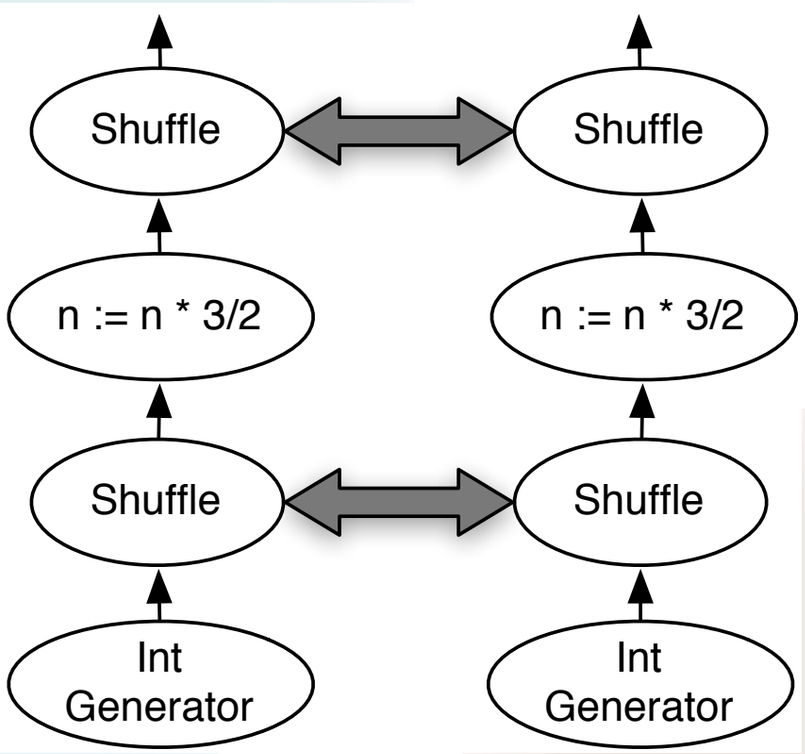
Implementation Decisions

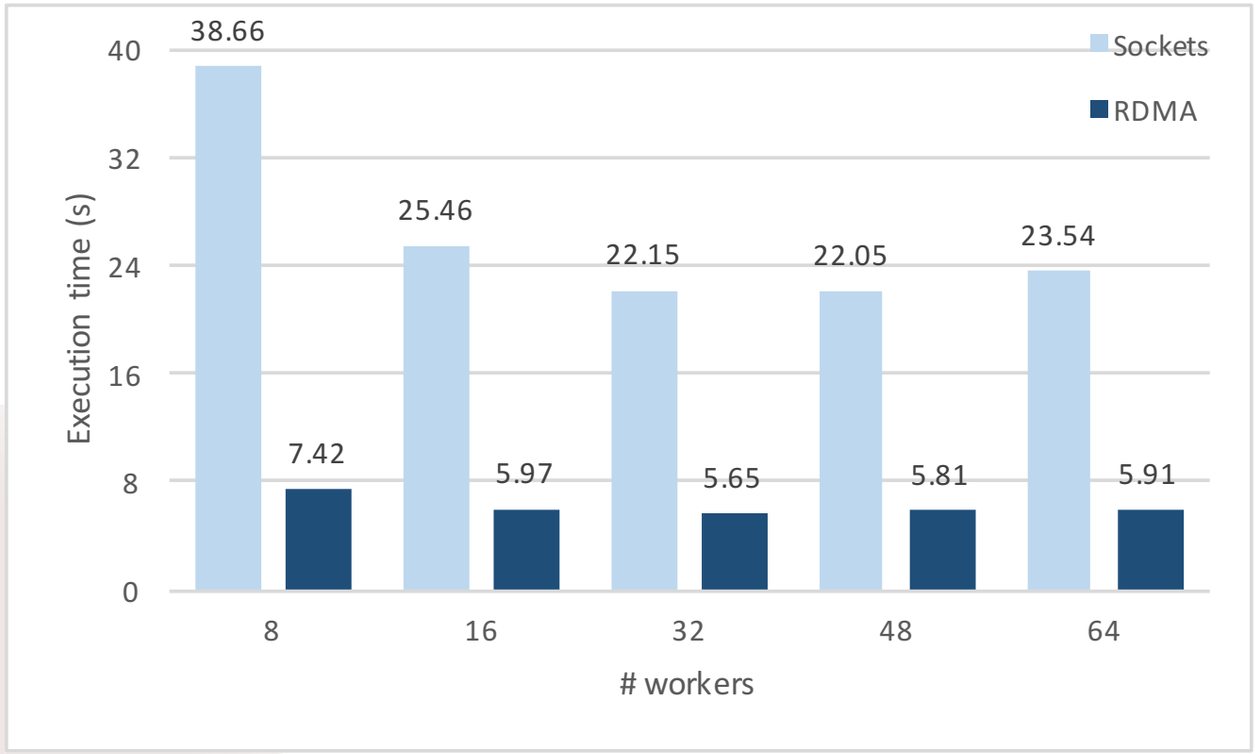
- Single connection between each pair of machines
  - Multiple workers share the same connection/ requests queue
- RDMA write (vs send/receive)
  - Avoids the need of a rendezvous protocol
  - Less CPU usage (remote side)

Optimizations

- Batch multiple rows/notifications before transferring data
  - Trades latency for reduced contention accessing network resources
- Pre-initialized data structures
  - Data structures initialized during creation of JDBC connection (reused for multiple queries)
  - Reduce overheads of JNI calls by using stateful verbs methods provided by jVerbs library

Performance Evaluation
Synthetic Query Plan

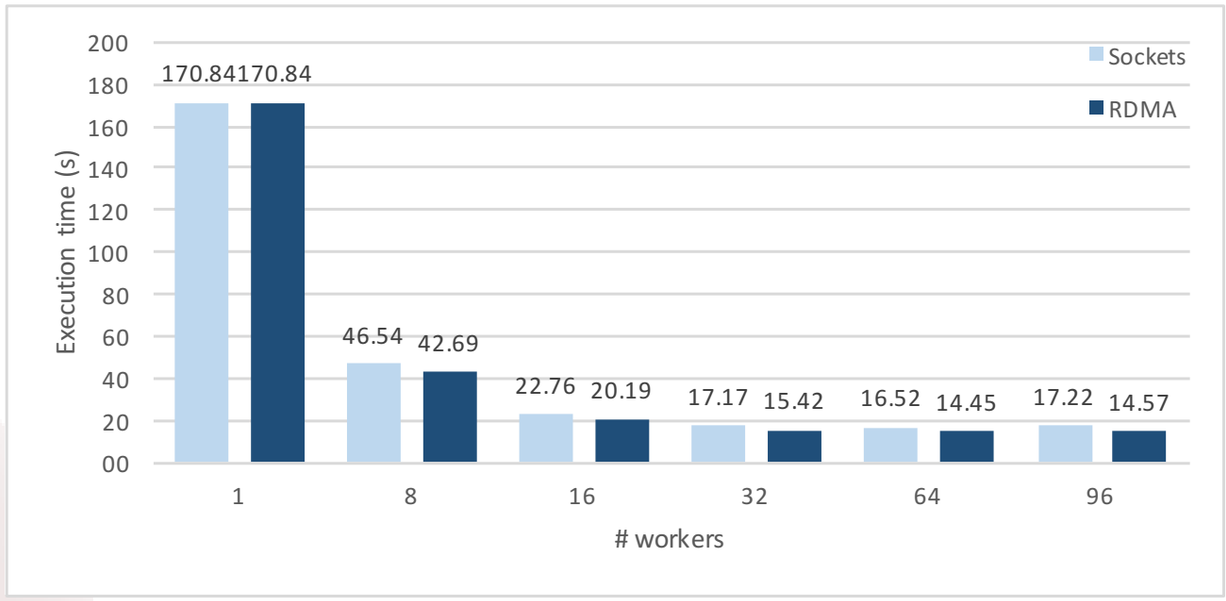




- 8 machines (Intel Core i3 CPU, 2 cores, 8GB RAM, 1 GigE)
- Query plan where each machine shuffles 5M integers twice
- RDMA middleware compared with a socket middleware
  - RDMA middleware: execution time ~3.9x faster

Performance Evaluation
SQL Join Query

```
select ol_o_id, ol_w_id, ol_d_id,
   sum(ol_amount) as revenue,
   o_entry_d
from order_line, orders, new_order, customer
where o_entry_d > timestamp('2013-07-01 0.00.00')
   and c_id = o_c_id and c_w_id = o_w_id
   and c_d_id = o_d_id and no_w_id = o_w_id
   and no_d_id = o_d_id and no_o_id = o_id
   and ol_w_id = o_w_id and ol_d_id = o_d_id
   and ol_o_id = o_id and c_state like 'A%'
group by ol_o_id, ol_w_id, ol_d_id, o_entry_d
having revenue > 80000.00
order by revenue desc, o_entry_d;
```

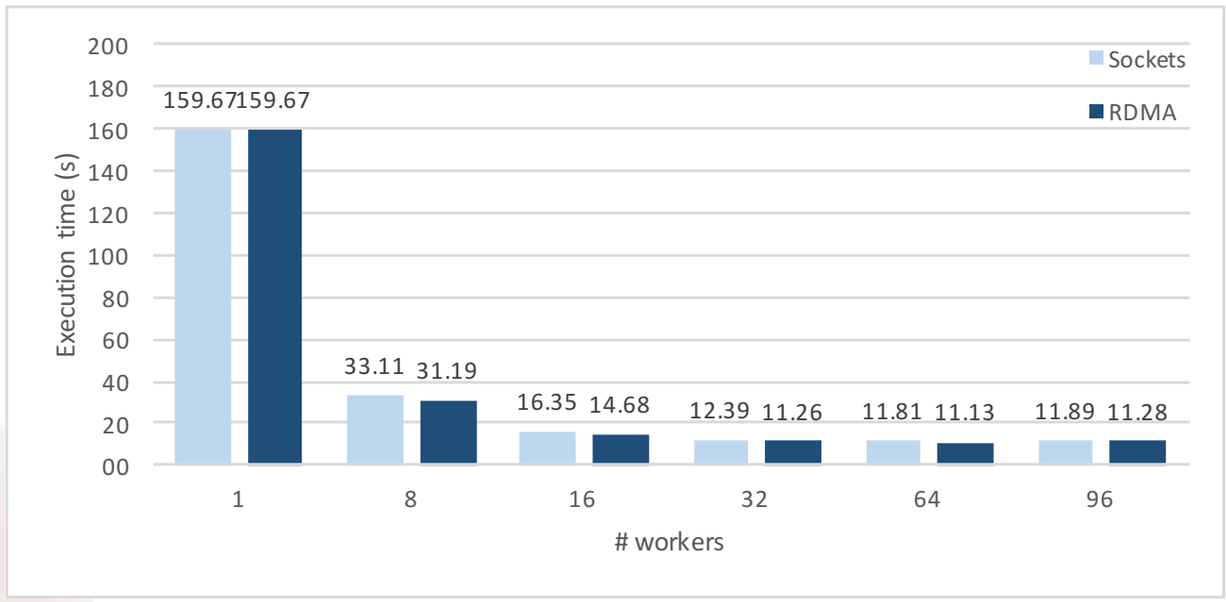


- Triple hash join with aggregation query
  - Uses 8 shuffle operators
- RDMA middleware: execution time ~1.14x faster

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

Performance Evaluation
SQL Aggregation Query

```
select ol_number, sum(ol_quantity) as sum_qty,
   sum(ol_amount) as sum_amount,
   sum(ol_quantity) / count(*) as avg_qty,
   sum(ol_amount) / count(*) as avg_amount,
   count(*) as count_order
from order_line
where ol_delivery_d > timestamp('2013-07-01 0:00:00')
group by ol_number
order by sum(ol_quantity) desc;
```



- Aggregation query
  - Uses 2 shuffle operators
- RDMA middleware: execution time around 1.06x faster

Summary

- Leverage from RDMA to improve performance of shuffling in large scale analytical processing
- Design an RDMA communication middleware to support push-based asynchronous shuffling
- Improves time spent on communication operations by 3.9x, when compared with a sockets-based middleware
- Using an RDMA approach is beneficial even when using software RDMA implementations