Request Window: an Approach to Improve Throughput of RDBMS-based Data Integration System by Utilizing Data Sharing Across Concurrent Distributed Queries

Rubao Lee, Minghong Zhou, Huaming Liao
lirubao@software.ict.ac.cn

Institute of Computing Technology
Chinese Academy of Sciences

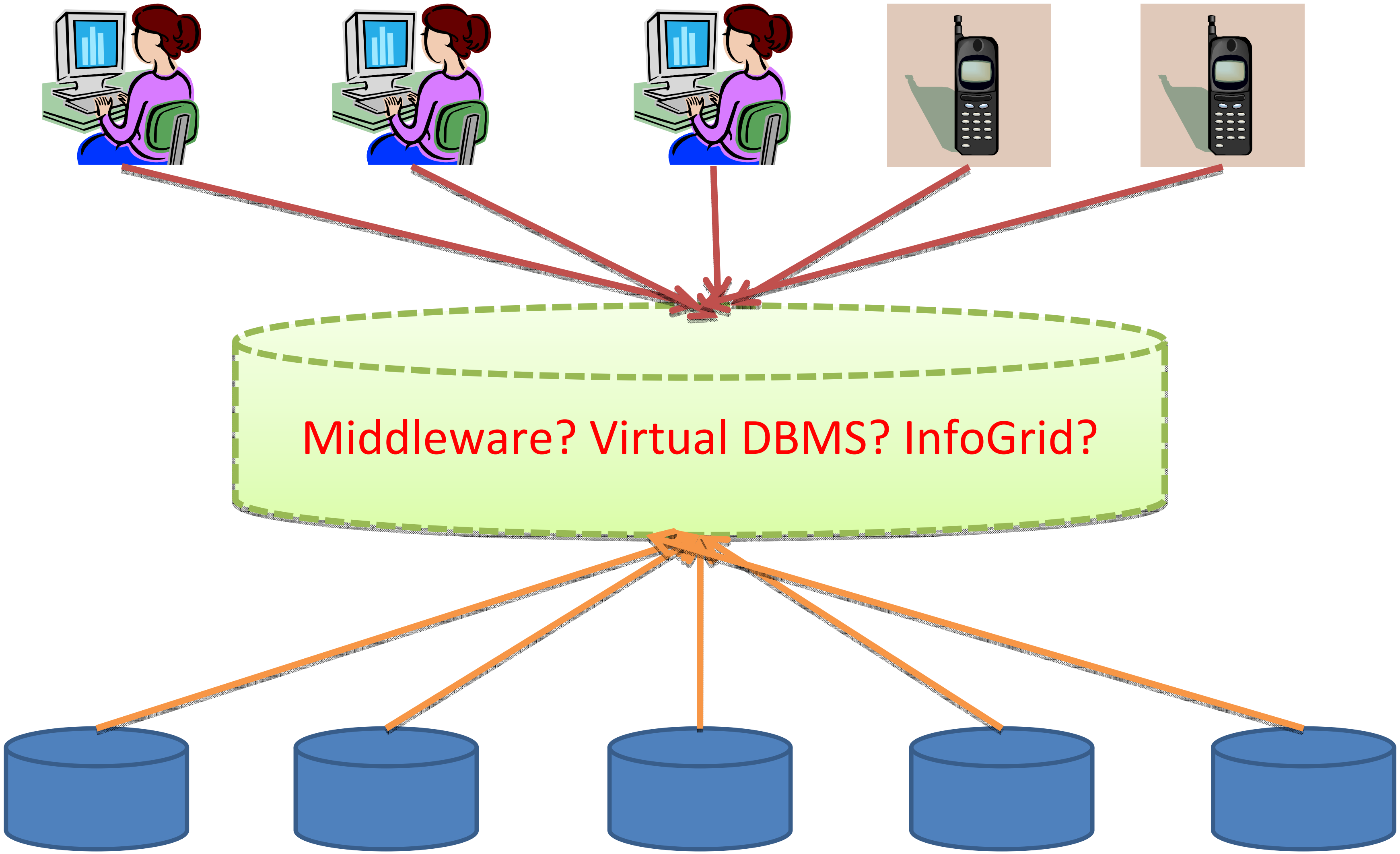
VLDB 2007

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

Outline

- Motivation: To Improve DQP Throughput
- Solution: Request Window
- Evaluation: Distributed TPC-H Queries
- Classification: Data Sharing Mechanisms
- Conclusion: Summary and Future Work

Typical Data Integration Service



From DBMS to Data Integration System

The Key: Distributed Query Processing (DQP)

New Leaf Node in Query Plan Tree

TableScan

Interacting with
storage devices

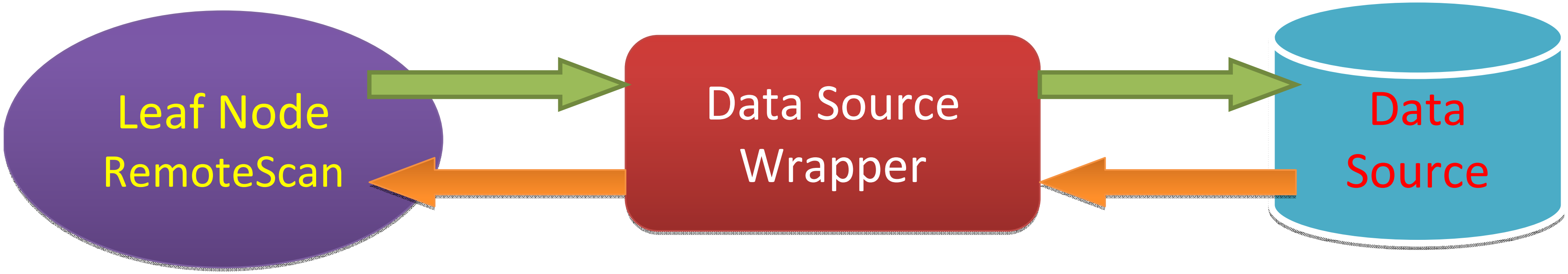
SeqScan,IndexScan,BitmapScan

→

RemoteScan

Interacting with
data source wrappers

Issuing data requests / fetching results



Leaf Node
RemoteScan

Data Source
Wrapper

Data
Source

IBM DB2 Information Integrator/ MS SQL Server 2005/ IGNITE

GOAL: Increase Overall DQP Throughput

- Only Consider how to execute a single query faster
  - Distributed Query Optimizer
  - New Join Algorithms
  - Adaptive Query Processing

NOT ENOUGH!

The key problem

How to execute multiple concurrent queries more efficiently ?

Data Sharing Is Important for DQP

Utilizing data sharing across concurrent queries to hide unnecessary I/O operations

Two factors of Distributed Query Processing

Network Speed, Source Burden

- ❑ Reducing unnecessary network transfers
- ❑ Reducing burdens of data sources

Data Sharing inside DBMS

- DBMS's query execution model:
  - One connection, one process
  - Execute each query in an independent process
  - Use a global buffer pool manager
- Foundation: Memory-Disk
  - Concurrent query processes can share disk pages!
  - Page Replacement Algorithm (LRU, ARC, 2Q, LIRS,…)

But, No Mem/Disk Hierarchy for DQP

- DQP inherits the underlying execution model
  - Independently executing each distributed query

  - But, no available buffer pool manager

- Data sources are not for random-access!
  - Issue a SQL and fetch a resultset (DBMS)
  - Issue a HTTP request and get a response (WebPage)
  - Issue a SOAP message and get a SOAP message (SOA)

No data sharing for DQP

- Each query execution process has to interact with data sources independently!

Redundant data requests issued to data sources

Redundant result data transferred over network

✗

The total throughput is limited

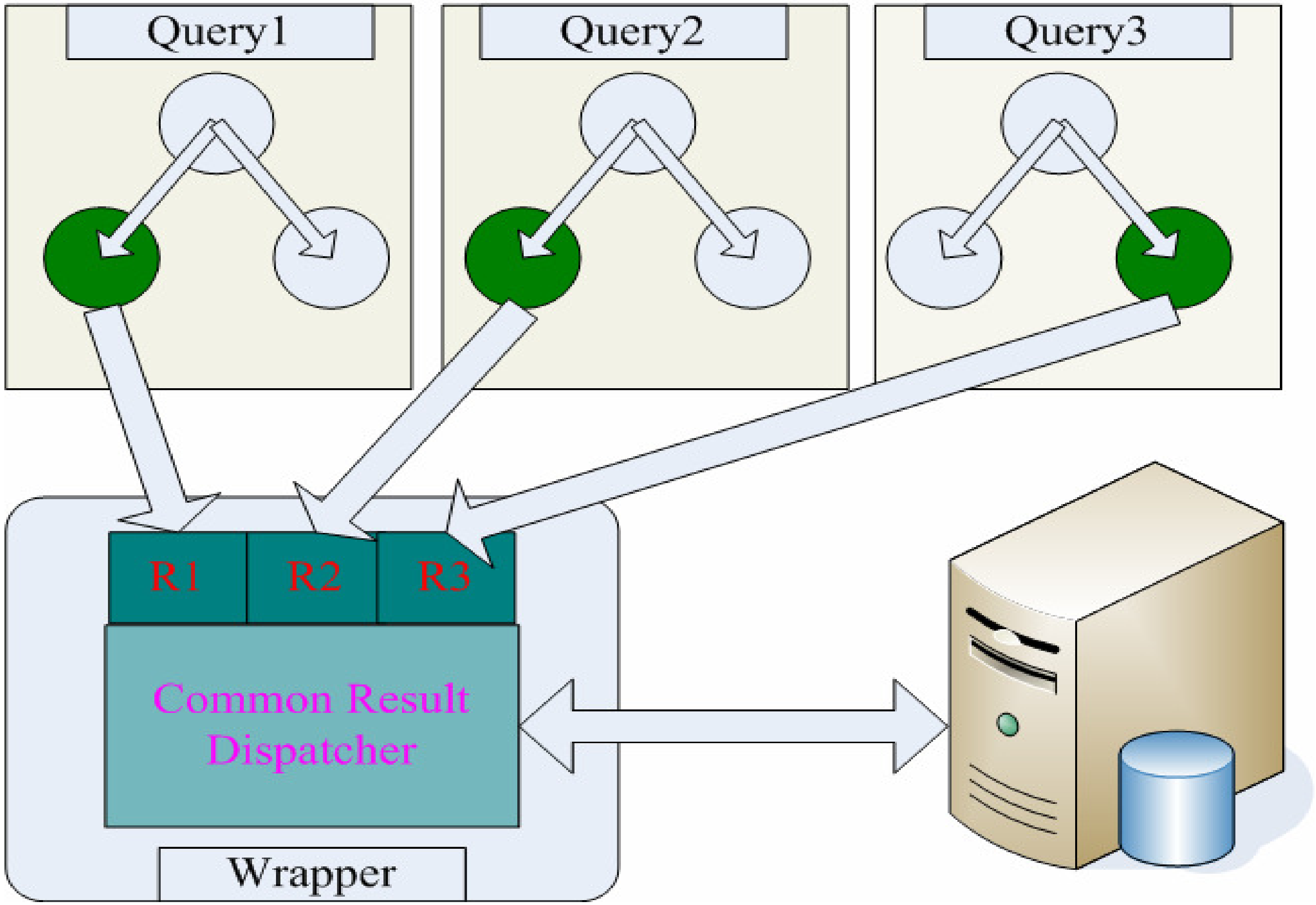
by network speed and computing power of sources!

Outline

- Motivation: To Improve DQP Throughput
- Solution: Request Window
- Evaluation: Distributed TPC-H Queries
- Classification: Data Sharing Mechanisms
- Conclusion: Summary and Future Work

Overview of Request Window

- Request Window: a batch-processing approach
  - Combining multiple data requests and dispatching results



Start-Fetch Wrapper

Foundation of Request Window

- Main idea: Decouple wrappers from query engine
  - A wrapper is in an independent process
  - Use IPC to connect wrappers and query engine
- Two Phases: by iterator model
  - Start: engine sends data request to wrapper (open)
  - Fetch: engine fetches result tuples from wrapper (next)

Two Benefits of Start-Fetch

- Parallelized query execution:
  - Wrappers can prefetch next tuples while query engine is consuming old tuples.
- The independent wrapper process can be a common place for multiple query engine processes.
  - The global buffer pool manager in DBMS!
  - Data sharing of multiple query processes can be possible!

What’s A Request Window?

- Each data request will be inserted into a corresponding waiting queue (a request window):
  - The data request will not be issued immediately

- **At a time**, the window will be issued:

  1: Combining all req into a common request:

  - *Select (c re (predict);*
  - *Gener e.*

  2: Sendin urce and receiving resultse

  3: Dispatching participating query engine process

When?

Window Size

Window Size: from window-creating to window-issuing!

How to determine the window size?

A large window size:
More data requests can be collected.

But, early requests have to wait! (unfair)

To Determine Window Size

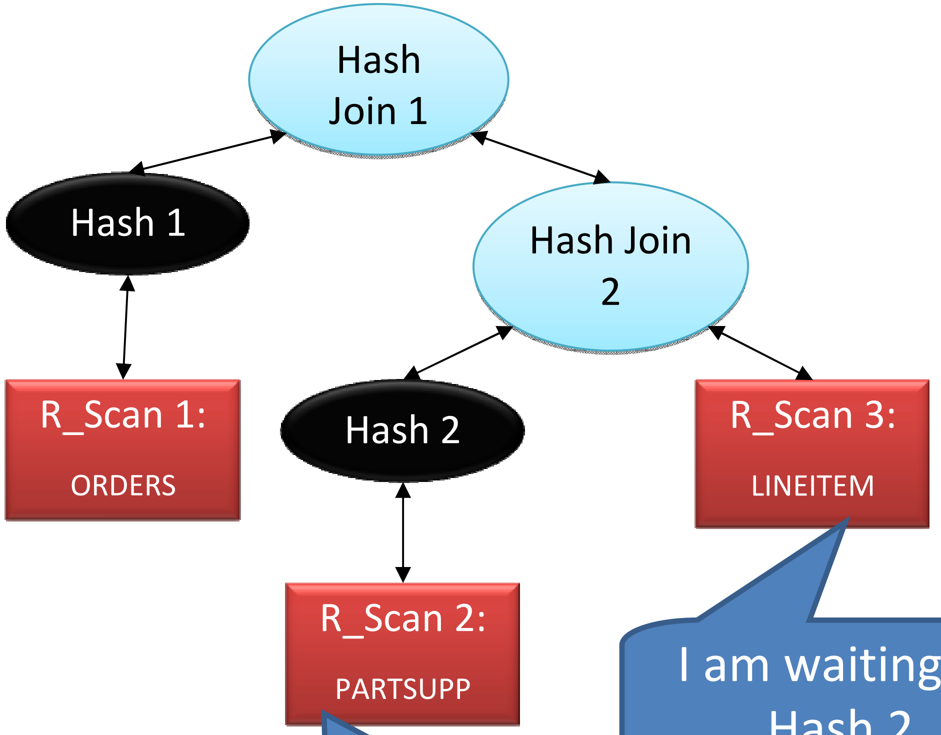
- DIOP: Delay Indicated by OPtimizer
  - Let the query optimizer indicate a tolerable delay time for each data request

- DAW: Dynamically Adjusting Window
  - Adjust the window size when a new data request arrives

DIOP: Why a request can be delayed?

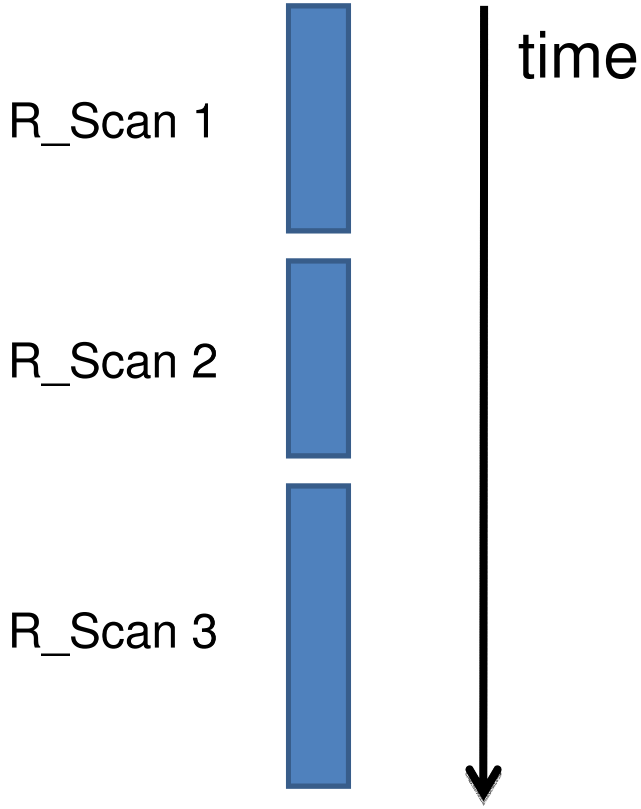
The iterator model : tuple fetching on demand!

ORDERS⋈ LINEITEM ⋈ PARTSUPP



I am waiting for Hash 2.

I am waiting for Hash 1.

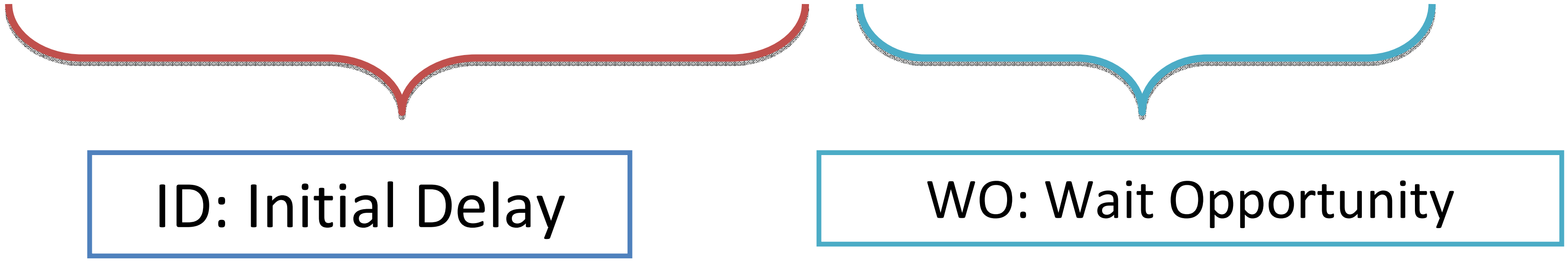


The execution can be divided into several phases

DIOP: How long a request can be delayed?

pipelined data fetch

Be ready for your tuples when I need them!



ID: Initial Delay

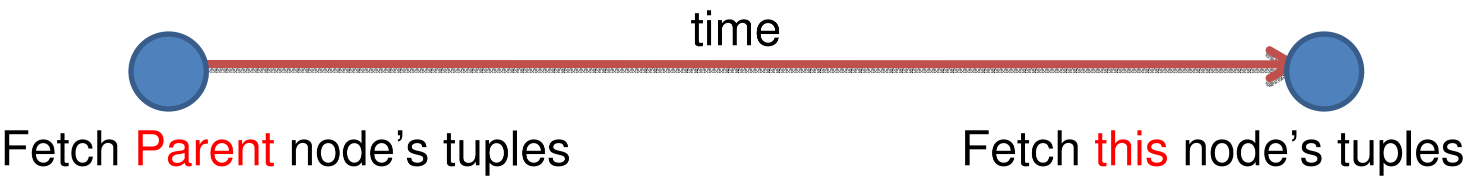
WO: Wait Opportunity

Maximized Delay Time of a request R generated by a leaf node N

$$MDT_R^N = WO_N - ID_R$$

DIOP: Algorithm-Related-Delay

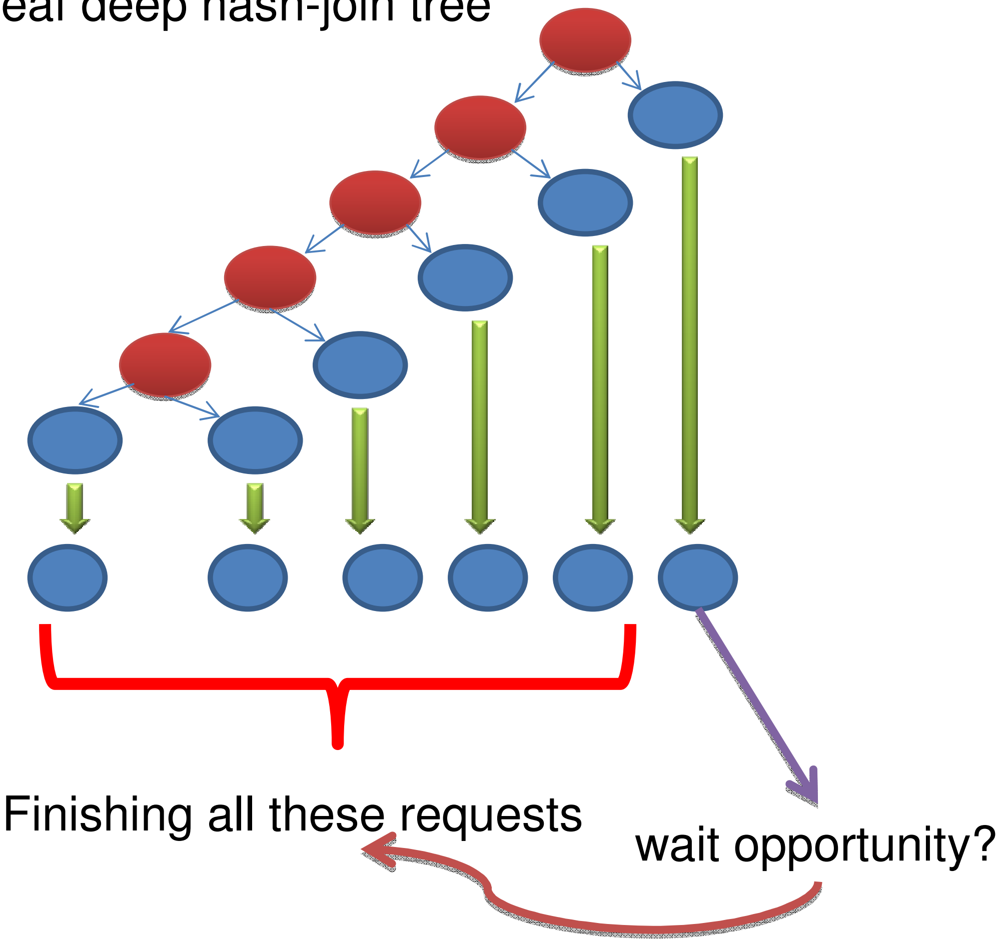
- "Wait Opportunity" of a node *N*
  - For non-root node: $WO_N = WO_P + ARD_N$
- ARD: *Algorithm Related Delay*



- Different relational operators have different ARDs
  - Hashjoin/Mergejoin
  - Union/intersection/difference

DIOP: Estimation for Hash-Join Tree

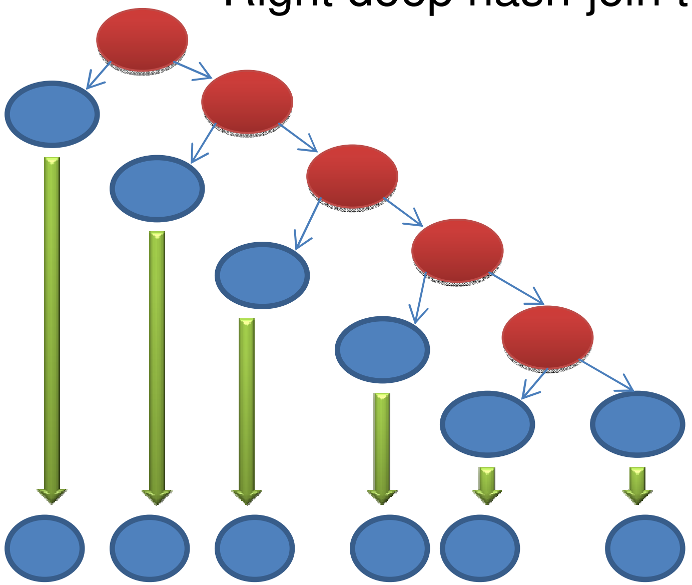
Leaf deep hash-join tree



Finishing all these requests

wait opportunity?

Right deep hash-join tree



Only consider time for data transfers over network

DAW: Dynamically Adjust Window

- Remember the goal: to determine window size
- DIOP is just the first step:
  - Each data request has an annotation of its maximized delay time
- A coordinator is required to determine the window size on the basis of delay times of all participating requests

Adjust window size when a new request arrives

DAW: Mechanism and Policy

- A background working-thread (wakes up : 1 second)
  - Resetting window size (if not ready)
  - Issuing window (if time out)
- Window Adjusting Policy (when a new request arrives)
  - Emergency-oriented policy

$$WS = MDT \;\; if\; MDT < WS$$

  - Throughput-oriented policy (DSS Queries)

$$WS = \frac{WS \times RC + MDT}{RC + 1} \quad if\; MDT < WS$$

WS: Window Size

MDT: Maximized Delay Time of the new request

RC: number of requests in the current window

The window size will never be increased!

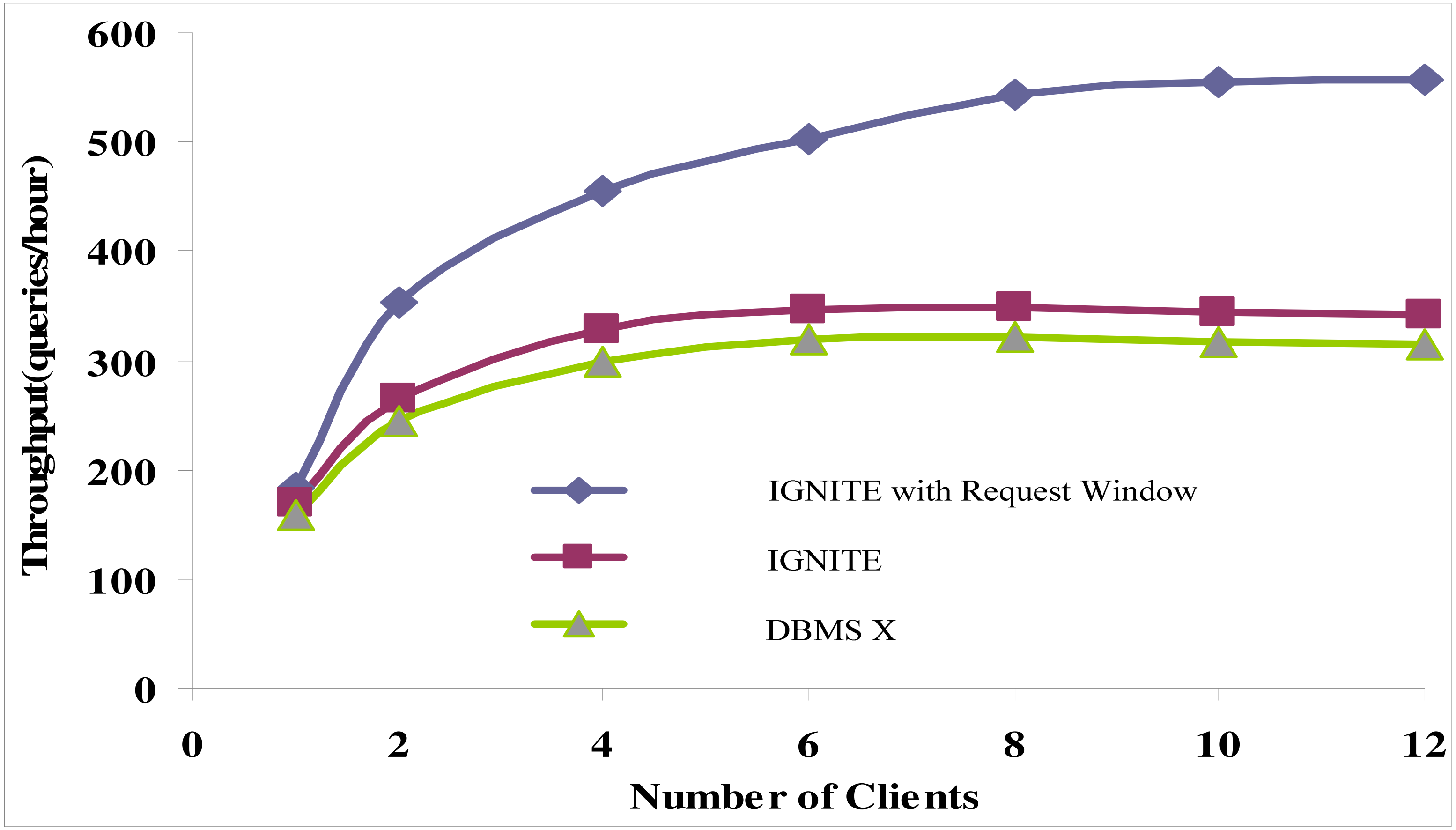
Outline

- Motivation: To Improve DQP Throughput
- Solution: Request Window
- Evaluation: Distributed TPC-H Queries
- Classification: Data Sharing Mechanisms
- Conclusion: Summary and Future Work

Experiments Setup

- IGNITE: on top of PostgreSQL
- TPC-H: 100MB (scale 0.1)
- IGNITE Machine:
  - Intel P4 Xeon 2.4GHz x4, 2GB Mem, Linux 2.4.18 SMP
- Data source Machines:
  - Intel P4 2.8GHz, 512MB Mem, Freebsd 5.4
  - PostgreSQL
  - Each TPC-H table is provided by a data source
- 100M LAN

Improvement of Overall Throughput



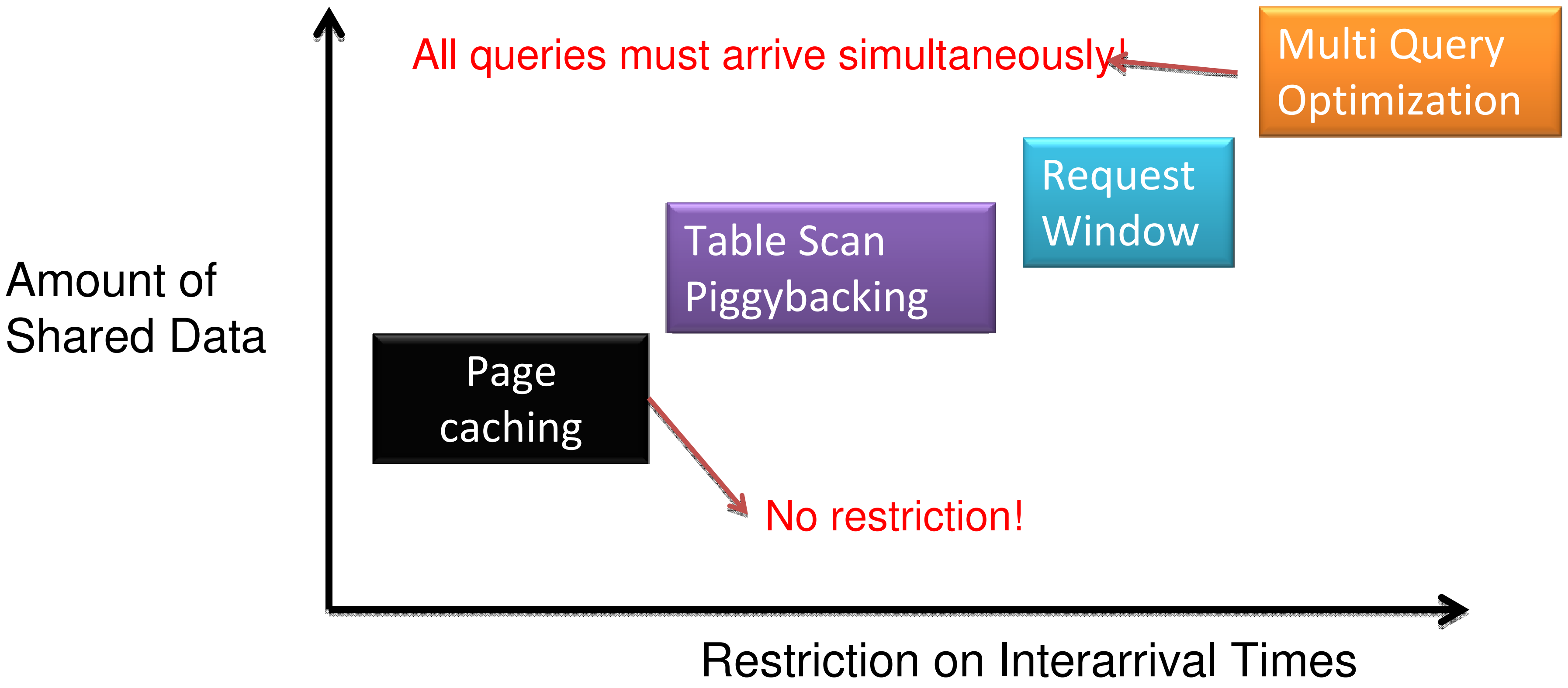
Up to a 1.7x speedup

Outline

- Motivation: To Improve DQP Throughput
- Solution: Request Window
- Evaluation: Distributed TPC-H Queries
- Classification: Data Sharing Mechanisms
- Conclusion: Summary and Future Work

Related Data Sharing Techniques

- Two Correlated Factors:
  - Restriction on interarrival times (deadline for sharing)
  - Amount of shared data (We can share data, but how much?)



This is a rough comparison!

Discussions and Future Work

- In a word:

Improve total throughput without sacrificing the response time of individual query execution

- Request Window is suitable for running concurrent DSS queries

- It is hard to make exactly estimation for delay opportunities
- Add Window Notification Mechanism
  - Monitoring query execution progress
  - Notifying wrapper to issue window

Thank You