Communication Cost for Parallel Query Processing

Paul Beame, Paraschos Koutris and Dan Suciu
University of Washington, Seattle, WA

Speaker: Yoav Nahshon

Seminar in Databases (Course No. 236826)
Spring 2017

Technion - Israel Institute of Technology
Computer Science Department

*Some slides were adapted from lectures by Dan Suciu

Beame, Koutris, Suciu, PODS’2013
Beame, Koutris, Suciu, PODS’2014
Balazinska, Chu, Suciu, SIGMOD’2015
Beame, Koutris, Suciu, ICDT’2016
Outline

Introduction

The MPC Model

Join Algorithms in the MPC Model

Other Parallel Models

Summary
Outline

Introduction

The MPC Model

Join Algorithms in the MPC Model

Other Parallel Models

Summary
Distribution introduces important concerns beyond those in the case of parallelism on a single multicore/multi-processor machine:

- Partial failure: crash failures of a subset of the machines involved in a distributed computation
- Latency: certain operations have a much higher latency than other operations due to network communication
## Latency Numbers

<table>
<thead>
<tr>
<th>Latency Description</th>
<th>ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>SSD random read</td>
<td>150,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Roundtrip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Read 1MB sequentially from SSD</td>
<td>1,000,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet US → Europe → US</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>

Original compilation by Jeff Dean & Peter Norvig, w/ contributions by Joe Hellerstein & Erik Meijer
## Latency Numbers \times 1\ Billion

<table>
<thead>
<tr>
<th>Activity</th>
<th>Latency (ns)</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000</td>
<td>50</td>
</tr>
<tr>
<td>Send 2K bytes over 1Gbps network</td>
<td>20,000</td>
<td>5.5</td>
</tr>
<tr>
<td>SSD random read</td>
<td>150,000</td>
<td>1.7</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
<td>2.9</td>
</tr>
<tr>
<td>Roundtrip within same datacenter</td>
<td>500,000</td>
<td>5.8</td>
</tr>
<tr>
<td>Read 1MB sequentially from SSD</td>
<td>1,000,000</td>
<td>11.6</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
<td>16.5</td>
</tr>
<tr>
<td>Read 1MB sequentially from disk</td>
<td>20,000,000</td>
<td>7.8</td>
</tr>
<tr>
<td>Send packet US → Europe → US</td>
<td>150,000,000</td>
<td>4.8</td>
</tr>
</tbody>
</table>

Original compilation by Jeff Dean & Peter Norvig, w/ contributions by Joe Hellerstein & Erik Meijer
<table>
<thead>
<tr>
<th>Latency Numbers × 1 Billion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 cache reference</strong></td>
</tr>
<tr>
<td><strong>Branch mispredict</strong></td>
</tr>
<tr>
<td><strong>L2 cache reference</strong></td>
</tr>
<tr>
<td><strong>Mutex lock/unlock</strong></td>
</tr>
<tr>
<td><strong>Main memory reference</strong></td>
</tr>
<tr>
<td><strong>Compress 1K bytes with Zippy</strong></td>
</tr>
<tr>
<td><strong>Send 2K bytes over 1Gbps network</strong></td>
</tr>
<tr>
<td><strong>SSD random read</strong></td>
</tr>
<tr>
<td><strong>Read 1 MB sequentially from memory</strong></td>
</tr>
<tr>
<td><strong>Roundtrip within same datacenter</strong></td>
</tr>
<tr>
<td><strong>Read 1MB sequentially from SSD</strong></td>
</tr>
<tr>
<td><strong>Disk seek</strong></td>
</tr>
<tr>
<td><strong>Read 1MB sequentially from disk</strong></td>
</tr>
<tr>
<td><strong>Send packet US → Europe → US</strong></td>
</tr>
</tbody>
</table>

Original compilation by Jeff Dean & Peter Norvig, w/ contributions by Joe Hellerstein & Erik Meijer
## Latency Numbers × 1 Billion

<table>
<thead>
<tr>
<th>Operation</th>
<th>L1 ns</th>
<th>L1 sec</th>
<th>L2 ns</th>
<th>L2 sec</th>
<th>Mutex lock/unlock ns</th>
<th>Mutex lock/unlock sec</th>
<th>Main memory reference ns</th>
<th>Main memory reference sec</th>
<th>Compress 1K bytes with Zippy ns</th>
<th>Compress 1K bytes with Zippy min</th>
<th>Send 2K bytes over 1Gbps network ns</th>
<th>Send 2K bytes over 1Gbps network hr</th>
<th>SSD random read ns</th>
<th>SSD random read days</th>
<th>Read 1 MB sequentially from memory ns</th>
<th>Read 1 MB sequentially from memory days</th>
<th>Roundtrip within same datacenter ns</th>
<th>Roundtrip within same datacenter days</th>
<th>Read 1MB sequentially from SSD ns</th>
<th>Read 1MB sequentially from SSD days</th>
<th>Disk seek ns</th>
<th>Disk seek weeks</th>
<th>Read 1MB sequentially from disk ns</th>
<th>Read 1MB sequentially from disk months</th>
<th>Send packet US → Europe → US ns</th>
<th>Send packet US → Europe → US years</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5ns</td>
<td>0.5 sec</td>
<td>7ns</td>
<td>7 sec</td>
<td>25ns</td>
<td>25 sec</td>
<td>100ns</td>
<td>100 sec</td>
<td>3,000ns</td>
<td>50 min</td>
<td>20,000ns</td>
<td>5.5 hr</td>
<td>150,000ns</td>
<td>1.7 days</td>
<td>250,000ns</td>
<td>2.9 days</td>
<td>500,000ns</td>
<td>5.8 days</td>
<td>1,000,000ns</td>
<td>11.6 days</td>
<td>10,000,000ns</td>
<td>16.5 weeks</td>
<td>20,000,000ns</td>
<td>7.8 months</td>
<td>150,000,000ns</td>
<td>4.8 years</td>
</tr>
</tbody>
</table>
# Latency Numbers

<table>
<thead>
<tr>
<th>Latency Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5ns</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5ns</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7ns</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25ns</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100ns</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000ns</td>
</tr>
<tr>
<td>Send 2K bytes over 1Gbps network</td>
<td>20,000ns</td>
</tr>
<tr>
<td>SSD random read</td>
<td>150,000ns</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000ns</td>
</tr>
<tr>
<td>Roundtrip within same datacenter</td>
<td>500,000ns</td>
</tr>
<tr>
<td>Read 1MB sequentially from SSD</td>
<td>1,000,000ns</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000ns</td>
</tr>
<tr>
<td>Read 1MB sequentially from disk</td>
<td>20,000,000ns</td>
</tr>
<tr>
<td>Send packet US → Europe → US</td>
<td>150,000,000ns</td>
</tr>
</tbody>
</table>

Original compilation by Jeff Dean & Peter Norvig, w/ contributions by Joe Hellerstein & Erik Meijer
Reducing Latency

Ways to reduce latency:

- Reduce disk I/O operations by keeping data in memory as much as possible
  - Major concept in Spark

- Reduce network traffic
  - Keep nodes in clusters geographically close
  - *Devise algorithms optimized for distributed settings* (better understanding of the system is required - this talk!)
Parallel Databases

Remember: goal is to improve performance by executing multiple operations in parallel

Settings:
- Input data of size \( m \) is partitioned on \( p \) servers, connected by a network
  - *Balanced partition*: Each server holds \( \approx m/p \) data
  - *Skewed partition*: Some server holds \( \gg m/p \) data
- Query processing involves local computation and communication

Usually, the input data is already partitioned, but we may need to re-partition for a particular problem (*data reshuffling*)
- Requires a global *synchronization* of all servers
- Skewed data leads to *stragglers*
  - “The curse of the last reducer” [SV11]
MapReduce Dataflow

Source: http://aimotion.blogspot.co.il/2012/08/introduction-to-recommendations-with.html
Measuring Performance

Performance parameters: [DG92]

*Speed-up*  Twice as much hardware can perform the task in half the elapsed time

*Scale-up*  Twice as much hardware can perform twice as large a task in the elapsed time

![Diagram showing speedup and batch scaleup with various hardware configurations and speed measurements](image)
We are interested in understating how data is processed in today’s distributed systems

- E.g., MapReduce, Hive, Spark, Tensorflow, etc.

Understanding $\Rightarrow$ formal (mathematical) model

Settings:
- Shared-nothing architecture
- Data is evenly distributed among machines (as in HDFS)
- Data reshuffling/synchronization
- New bottleneck: communication

In the following slides:
- The MPC model
- Analysis of join algorithms in the MPC model
Outline

Introduction

The MPC Model

Join Algorithms in the MPC Model

Other Parallel Models

Summary
Massively Parallel Communication Model (MPC)

[BKS13] The MPC model is the following:

- **$p$ servers** are connected by a network (∼thousands)
  - Servers are infinitely powerful
- **Input data** of size $m$ (∼TBs)
  - Initially each server has $\frac{m}{p}$ data, evenly distributed
- Computation is performed in **rounds**, each consists of:
  1. Communication step: servers exchange data
  2. Computation step: servers perform local computations
- The **only** cost is communication, defined by the load (will be defined shortly), and number of rounds
- At the end of each round the output result is present in the union of the $p$ machines
Execution in MPC
Load

Definition (The Load of an algorithm on the MPC Model)

$L_u = \max \text{ amount of data received by server } u \text{ during any round}$

$L = \max_u L_u$

Speed: $V(p) = 1/L(p)$

Speedup: $S(p) = L(1)/L(p)$

Goal: Analyze tradeoff between number of rounds and amount of communication
Naive 1-Round: Send entire data to server 1, compute locally; 
\[ L = m \]

Naive p-Rounds: At each round, send a \( \frac{m}{p} \)-fragment of the data to server 1, then compute locally; \( L = \frac{m}{p} \)

Ideal Algorithms: 1-Round, load \( L = \frac{m}{p} \) (but rarely possible)

Real Algorithms: \( O(1) \) rounds, and \( L = O\left(\frac{m}{p^{1-\varepsilon}}\right) \), for \( 0 \leq \varepsilon < 1 \)
Establishment of lower and upper bounds in the MPC model for computing a *full conjunctive query*, in three different settings:

1. Single round, skew-free data
2. Single round, skewed data
3. Multiple rounds, skew-free data

Today’s talk: one round, skew-free data
Outline

Introduction

The MPC Model

Join Algorithms in the MPC Model

Other Parallel Models

Summary
Background on Databases

- Vocabulary: \( R_1, R_2, \ldots, R_k \), arities: \( r_1, r_2, \ldots, r_k \)
- Domain size: \( n \)
- Database instance: \( D = (R^D_1, R^D_2, \ldots, R^D_k) \), \( R^D_j \subseteq [n]^{r_j}, n > 0 \)
- Input cardinalities: \( m_1 = |R^D_1|, \ldots, m_k = |R^D_k| \)
- Full conjunctive query, a.k.a. natural multi-join:
  \[
  Q(x_1, \ldots, x_l) = R_1(x_1), R_2(x_2), \ldots, R_k(x_k)
  \]
  Query is full if every variable in the body appears in the head
  - For example, \( Q(x) = R(x, y) \) is not full
- \( Q^D := \text{the answer on } D \)
Join - Simple Example

\[ Q(x, y, z) = R(x, y), S(y, z) \quad \left( \equiv R \bowtie S \right) \]

\[
\begin{array}{|c|c|}
\hline
R^D & x & y \\
\hline
5 & 1 \\
3 & 1 \\
3 & 2 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
S^D & y & z \\
\hline
1 & 6 \\
1 & 8 \\
2 & 8 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|}
\hline
Q^D & x & y & z \\
\hline
5 & 1 & 6 \\
5 & 1 & 8 \\
3 & 1 & 6 \\
5 & 1 & 8 \\
3 & 2 & 6 \\
\hline
\end{array}
\]

Note: \[ 0 \leq |Q^D| \leq |R^D| \times |S^D| \]
Key Question
Given a query $Q$ on a large database, how much communication do we need to perform in order to compute $Q$ distributively, on $p$ servers?

We begin our analysis with a simple case:

$$Q(x, y, z) = R(x, y), S(y, z)$$
Hash Join

\[ Q(x, y, z) = R(x, y), S(y, z) \quad |R^D| = m_R, \quad |S^D| = m_S \]

Choose a random hash function \( h \) that maps attribute values to \([p]\) (= \{1, \ldots, p\})

**Round 1: Communication**
In each server:
- send every tuple \((a, b) \in R^D\) to server \( h(b) \)
- send every tuple \((c, d) \in S^D\) to server \( h(c) \)

**Round 1: Computation**
In each server:
- compute the join on the local fragments of \( R^D, S^D \)

Load:
\[
m_R/p + m_S/p. \quad \text{If } m := m_R + m_S, \text{ then } L = m/p \quad \text{linear speedup}
\]
- Assuming no skewed values and \( h \) is a “good” hash function
Broadcast Join

\[ Q(x, y, z) = R(x, y), S(y, z) \quad |R^D| = m_R \gg |S^D| = m_S \]

**Round 1: Communication**
Every server broadcasts all its local \( S \)-tuples to all other servers

**Round 1: Computation**
In parallel, each server \( u \) computes the join \( R_u(x, y) \bowtie S(y, z) \) of its local fragment \( R_u^D \) with \( S^D \)

Load: \( m_S \) (even if data is skewed)

\[ m_S \leq \frac{m_S}{p} + \frac{m_R}{p} \quad \Rightarrow \quad m_S \leq \frac{m_R}{p-1} \]

Used a lot in practice (e.g., Pig)
Cartesian Join

\[ Q(x, y, z) = R(x, y), S(y, z) \]

When \( R^D \) and \( S^D \) have a single \( y \)-value then effectively we compute \( R(x) \times S(z) \) on one machine; \( L = m \)

Efficient algorithm:
Place the \( p \) servers in a \( p_x \times p_z \) rectangle, where \( p = p_x \cdot p_z \)
  ▶ The numbers \( p_x \) and \( p_z \) are called \textit{shares}

Let \( h_1, h_2 \) be independently chosen random hash functions

\textbf{Round 1:} In parallel, each server does the following:
- Send \( R(v) \) to all servers \((h_1(v), *)\)
- Send \( S(w) \) to all servers \((*, h_2(w))\)

Then compute \( Q \) locally

\textbf{Correctness:} answer \((v, w)\) is in machine \((h_1(v), h_2(w))\)
Cartesian Product - Load Analysis

- $R$ is partitioned into $p_x$ buckets
- Then, each machine receives $O\left(\frac{m_R}{p_x}\right)$ tuples from $R$
- Similarly, $O\left(\frac{m_S}{p_z}\right)$ from $S$

Therefore, the load is:

$$L = \frac{m_R}{p_x} + \frac{m_S}{p_z} \geq 2 \left(\frac{m_R m_S}{p_x p_z}\right)^{1/2} = 2 \left(\frac{m_R m_S}{p}\right)^{1/2}$$

First inequality is due to $a + b \geq 2\sqrt{ab}$ \hspace{1cm} \text{proof in Extra Slides}

Optimal values: $p_x = \left(\frac{m_R}{m_S} p\right)^{1/2}$, $p_z = \left(\frac{m_S}{m_R} p\right)^{1/2}$

Observation: when $m_R = m_S$ then $p_x = p_z = \sqrt{p}$
Claim
$L = \Omega \left( 2 \left( \frac{m_R m_S}{p} \right)^{1/2} \right)$

I.e., the previously shown allocation of shares is optimal

Proof.

▶ Suppose machine $j$ receives $m_{R,j}, m_{S,j}$ tuples from $R$ and $S$ respectively

▶ Then, the number of tuples it can output is at most

$$m_{R,j} m_{S,j} \leq (m_{R,j} + m_{S,j})^2 / 4 = L_j^2 / 4$$

▶ The total output $\sum_j L_j^2 / 4$ must be at least $m_R m_S$

Therefore:

$$m_R m_S \leq \sum_j L_j^2 / 4 \leq \sum_j L_j^2 / 4 = pL^2 / 4$$

▶ Thus: $L \geq 2 \left( \frac{m_R m_S}{p} \right)^{1/2}$
The Triangle Query

\[ Q(x, y, z) = R(x, y), S(y, z), T(z, x) \]

Applications: social networks, biological motifs, graph databases

**Round 1:**
Hash-partitioned join: \( Aux(x, y, z) = R(x, y), S(y, z) \)

**Round 2:**
Hash-partitioned join: \( Q(x, y, z) = Aux(x, y, z), T(z, x) \)

Load: can be as high as \( m^2/p \) because of the intermediate result!

Can we compute triangles with a smaller load?
The triangle query can be computed in one round!

- Algorithm introduced by Afrati & Ullman [AU10]
  - A.k.a. *Shares Algorithm*
  - For MapReduce

- Analyzed/optimized in [BKS13, BKS14]
  - HyperCube Algorithm
  - For the MPC model
The HyperCube Algorithm

\[ Q(x, y, z) = R(x, y), S(y, z), T(z, x) \]

Place the \( p \) servers in a \( p_x \times p_y \times p_z \) cube, where \( p = p_x \cdot p_y \cdot p_z \)

Let \( h_1, h_2, h_3 \) be independently chosen random hash functions

**Round 1:** In parallel, each server does the following:

- Send \( R(x, y) \) to all servers \((h_1(x), h_2(y), \ast)\)
- Send \( S(y, z) \) to all servers \((\ast, h_2(y), h_3(z))\)
- Send \( T(z, x) \) to all servers \((h_1(x), \ast, h_3(z))\)

Then compute \( Q \) locally

**Correctness:** triangle \( \Delta(a, b, c) \) is in machine \((h_1(a), h_2(b), h_3(c))\)
Load Analysis I

- Let us focus on how $R$ is distributed
- Observation: after 1\textsuperscript{st} round $R$ is partitioned into $p_x \cdot p_y$ buckets
- The size each bucket is approximately the same
  - assuming $R$ is skew-free
- Thus the expected load is $L_R = O(m_R / (p_x p_y))$
- Similarly, $L_S = O(m_S / (p_y p_z))$ and $L_T = O(m_T / (p_x p_z))$

To find the total load $L$ we construct an optimization problem

Objective: minimize $L$

Constraints: $L_R \geq m_R / (p_x p_y)$ (similarly for $S$ and $T$)

$p_x \cdot p_y \cdot p_z \leq p$
Load Analysis II

Objective: minimize $L$

Constraints: $L_R \geq m_R/(p_x p_y)$ (similarly for $S$ and $T$)

$\forall p, x, y, z$

We now:

- **take** logarithms with base $p$ on both sides
- **denote:** $\lambda = \log_p L$, $e_i = \log_p p_i$ for $i \in \{x, y, z\}$
- **transform** the above into a linear program

\[
\begin{align*}
\text{minimize} & \quad \lambda \\
\text{subject to} & \quad e_x + e_y + \lambda \geq \log_p m_R \\
& \quad e_y + e_z + \lambda \geq \log_p m_S \\
& \quad e_x + e_z + \lambda \geq \log_p m_T \\
& \quad e_x + e_y + e_z \leq 1 \\
& \quad e_x, e_y, e_z \geq 0
\end{align*}
\]
By solving the linear program of the last slide we have that

\[ L = O \left( \max \left\{ \frac{m_R}{p}, \frac{m_S}{p}, \frac{m_T}{p}, \frac{(m_R m_S m_T)^{1/3}}{p^{2/3}} \right\} \right) \]

If \( m_R = m_S = m_T = m \) then:

- Shares are equal: \( p_x = p_y = p_z = p^{1/3} \)
- \( L = O(m/p^{2/3}) \)

Thus, speedup is not linear

Can we compute Triangles with \( L = m/p \)? \textbf{No!}

\textbf{Theorem [BKS13]}

Any 1-round algorithm has load \( L = \Omega(m/p^{2/3}) \), even on inputs with no skew
So far we discussed:
  - Join \( L = \frac{m}{p} \)
  - Triangles \( L = \frac{m}{p^{2/3}} \)

For full conjunctive queries the authors establish tight upper and lower bounds on the maximum load.

When input has no skew, join queries can be computed optimally using the general HyperCube Algorithm.
The General HyperCube Algorithm

\[ q(x_1, \ldots, x_k) = S_1(x_1), \ldots, S_l(x_l) \]

Write: \( p = p_1 \times \ldots \times p_k \)  \textit{shares}

\textbf{Round 1:} In parallel, each server does the following:

- Send \( S(x_{j_1}, x_{j_2}, \ldots) \) to all servers whose coordinates agree with \( h_{j_1}(x_{j_1}), h_{j_2}(x_{j_2}), \ldots \)
  - Broadcast along the dimensions that are missing
- Then compute \( Q \) locally

\[ \text{More details in Extra Slides} \]
Main Result: Single Round, Skew-Free Data

\[ Q(x) = R_1(x_1), \ldots, R_k(x_k) \quad m_1 = |R_1^D|, \ldots, m_k = |R_k^D| \]

**Theorem**

Any algorithm that correctly computes \( Q^D \) on a skew-free database requires a load:

\[ L \geq \text{some geometric mean of } m_1, \ldots, m_k \]

\[ p^{1/u_0} \]

where \( u_0 \geq 1 \) depends on the query and the ratios of \( m_1, \ldots, m_k \)

Hence: speedup = \( p^{1/u_0} \) is sub-linear when \( u_0 > 1 \)

1. Upper bound: the HyperCube algorithm
2. Lower bound: a proof based on information theory
Other Models for Shared-Nothing Architectures

- **BSP** [Val90] - similar to the MPC model
  - Main difference: MPC does not consider computation cost

- **MRC** [KSV10] - Capture computations in the MapReduce framework
  - Does not consider problems related to query processing, but tasks such as sorting or the Minimum Spanning Tree problem
  - Provide no lower bounds on the communication or round complexity

- **Afrati-Ullman** [SASU13] - a model for MapReduce
  - Main parameter is the number of input tuples a reducer can receive (reducer size), while MPC takes the number of servers as an explicit parameter
  - No analysis on multi-round MapReduce algorithms
Outline

Introduction

The MPC Model

Join Algorithms in the MPC Model

Other Parallel Models

Summary
Communication is a new and the most significant bottleneck

There is a tradeoff between the number of rounds and maximum load for query processing tasks

The MPC model captures communication cost of modern distributed data analytics systems (MapReduce, Spark) and enables to better understand this tradeoff

This equips system designers with knowledge about:

- How much synchronization, communication and load the computation of a query requires
- What is possible to achieve under specific system constraints
The focus of this work: study of the class of full conjunctive queries in the MPC model
  - Central in processing relational data

HyperCube - optimal algorithm (w.r.t. lower bounds) for computing any query in one round

Follow-up research: skew and multiple rounds

Future work:
  - Beyond joins (e.g., queries with projections, unions and aggregation)
  - Asynchronous systems (e.g., GraphLab, Myria)


Online tutorial by Dan Suciu:
https://youtu.be/ATvW_S0bazk
Extra Slides
MapReduce Example - Word Count

Input: Deer Bear River, Car Car River, Deer Car Bear

Splitting: Deer Bear River, Car Car River, Deer Car Bear

Mapping:
- Deer, 1
- Bear, 1
- River, 1
- Car, 1
- Car, 1
- Car, 1
- Deer, 1
- Deer, 1
- River, 1
- River, 1

Shuffling:
- Bear, 1
- Bear, 1
- Car, 1
- Car, 1
- Car, 1
- Deer, 1
- Deer, 1
- River, 1
- River, 1

Reducing:
- Bear, 2
- Car, 3
- Deer, 2
- River, 2

Final result: Bear, 2
- Car, 3
- Deer, 2
- River, 2
Lemma

Let \(a\) and \(b\) be positive real numbers. Then:

1. \(\sqrt{ab} \leq \frac{a+b}{2}\); and
2. equality occurs iff \(a = b\)

Proof.

1. \((a + b)^2 - 4ab = (a - b)^2 \geq 0\); Therefore
2. \((a + b)^2 \geq 4ab\); or
3. \(\frac{a+b}{2} \geq \sqrt{ab}\).

4. If \(a = b\) then \(\sqrt{ab} = \frac{a+b}{2}\)
5. If \(\sqrt{ab} = \frac{a+b}{2}\) then, according to line 1, \((a - b)^2 = 0\), and so \(a = b\)
The General HyperCube Algorithm

\[ q(x_1, \ldots, x_k) = S_1(x_1), \ldots, S_l(x_l) \]

- Assign to each variable \( x_i, i = 1, \ldots, k \), a share \( p_i \) such that \( \prod_i^k p_i = p \)
- Denote: \( \mathcal{P} = [p_1] \times \cdots \times [p_k] \)
- Choose \( k \) independently chosen hash functions \( h_i : U \rightarrow [p_i] \)

**Round 1: Communication**
Send each tuple \( t \) of relation \( S_j(x_{i_1}, \ldots, x_{i_a}) \) to all machines in the destination subcube of \( t \):

\[ \mathcal{D}(t) = \{ y \in \mathcal{P} \mid \forall m \in [a] : h_{i_m}(t[i_m]) = y_{i_m} \} \]

**Round 1: Computation**
Compute \( Q \) locally
Fractional Edge Packing

\[ q(x_1, \ldots, x_k) = S_1(x_1), \ldots, S_l(x_l) \]

Fractional Edge Packing

A fractional edge packing of a query \( q \) is any feasible solution \( u = (u_1, \ldots, u_k) \) of the following linear constraints:

\[
\forall i \in [k] : \sum_{j : x_i \in \text{vars}(S_j)} u_j \leq 1 \\
\forall j \in [l] : u_j \geq 0
\]

Example: \( Q(x_1, x_2, x_3, x_4) = S_1(x_1, x_2), S_2(x_2, x_3), S_3(x_3, x_4) \)

An edge packing is a solution to:

\[
u_1 \leq 1, \quad u_1 + u_2 \leq 1, \quad u_2 + u_3 \leq 1, \quad u_3 \leq 1\]
Lower and Upper Bounds

\[ q(x_1, \ldots, x_k) = S_1(x_1), \ldots, S_l(x_l) \]

minimize \[ \lambda \]
subject to
\[ \sum_{i \in [k]} -e_i \geq -1 \]
\[ \forall j \in [l]: \sum_{x_i \in S_j} e_i + \lambda \geq \log_p m_j \]
\[ \forall i \in [k]: e_i \geq 0, \quad \lambda > 0 \]

Theorem: Lower [BKS14] and Upper [BKS13] Bounds

\[ L = \theta \left( \max_{u \in \text{pk}(q)} \left( \frac{\prod_{j=1}^{k} m_j^{u_j}}{p} \right)^{1/\sum_j u_j} \right) \]

where \( \text{pk}(q) \) is the set of all edge packings for \( q \)