Multi-join Query Evaluation – Outline

Part 1 Optimal Sequential Algorithms.

Part 2 Lower bounds for Parallel Algorithms.

Part 3 Optimal Parallel Algorithms.

Part 3 Data Skew.
Summary so far

\[ Q(x) = R_1(x_1), \ldots, R_\ell(x_\ell) \]

\[ |R_1| = m_1, \ldots, |R_\ell| = m_\ell \]

Sequential World  Cost: output size of \( Q \)

Upper bound \( m^{\rho^*} \); general: \( m_1^{u_1} \cdots m_\ell^{u_\ell} \). Fractional edge cover.

Lower bound (tightness): fractional vertex packing

Generic-join algorithm.

Parallel World  Cost: communication.

1-round, skew-free, equal-cardinalities.

Lower bound \( m/p^{1/\tau^*} \); general: \( (m_1^{u_1} \cdots m_\ell^{u_\ell} / p)^{1/\sum u_j} \) Fractional edge packing.

Upper bound: fractional vertex cover.

HyperCube algorithm.
Outline of Lecture 4

- Background: Hash-Based Partition
- Coping with Skew
- Multi-rounds (very short)
- Open Problems

Will consider only databases without skew
Understanding Skew (Review from Lecture 2)

Given: \( R(x, y) \) with \( m \) items, \( p \) servers.
Send each tuple \( R(x, y) \) to server \( h(y) \), where \( h = \text{random function} \).

**Claim 1** For each server \( u \), its expected load is \( E[L_u] = \frac{m}{p} \). Why?

**Claim 2** We say that \( R \) is *skewed* if some value \( y \) occurs more than \( \frac{m}{p} \) times in \( R \). Then some server has load \( > \frac{m}{p} \).

**Claim 3** If \( R \) is not skewed, the maximum load of all servers is \( O(\frac{m}{p}) \) with high probability. (Details in lecture 4.)

Take-away: we assume no skew, meaning every frequency is \( \leq \frac{m}{p} \), then:

\[
\text{Max-load} = O(\text{Expected load})
\]
Hash-Based Partition

- A *hash function* is a function from some domain $D$ to a range of integers $[p]$. E.g. $h : \text{char}(30) \rightarrow \{1, \ldots, p\}$

- A *random family of hash functions* is a set of hash functions, from which we select one at random.

- A *strongly universal* family of hash function is a set with the property that for any distinct values $x_1, \ldots, x_n \in D$, and any outputs $(u_1, \ldots, u_n) \in [p]^n$,

  $$\mathbb{P}(h(x_1) = u_1 \wedge \ldots \wedge h(x_n) = u_n) = \frac{1}{p^n}$$
Hash-Based Partition

Let $R$ be a bag (“multi-set”) with $m$ elements.

Partition $R$ into $p$ bins, using a hash function $h$: send element $x \in R$ to bin $h(x) \in [p]$. 

Denote $L_u = \text{number of elements in bin } u \in [p]$.

Example: $h(x) = [(x - \prime b) \mod 3] + 1$

- $x : a a a b c c d e e e$
- $h(x) : 3 3 3 1 2 2 3 1 1 1$
- $L_1 = 4$, $L_2 = 2$, $L_3 = 4$

Q: What is $E[L_u]$, for a fixed $u$?

A: $E[L_u] = \frac{m}{p}$

Q: What is $E[\max_u L_u]$?

A: May be as large as $m$. 

Dan Suciu

Multi-Joins – Lecture 4

March, 2015 7 / 29
Hash-Based Partition

Let $R$ be a bag ("multi-set") with $m$ elements.

Partition $R$ into $p$ bins, using a hash function $h$: send element $x \in R$ to bin $h(x) \in [p]$.

Denote $L_u =$ number of elements in bin $u \in [p]$ (a random variable).

Example: $h(x) = [(x - b') \mod 3] + 1$

```
x: a a a b c c d e e e e
h(x): 3 3 3 1 2 2 3 1 1 1 1
```
Hash-Based Partition

Let $R$ be a bag ("multi-set") with $m$ elements.

Partition $R$ into $p$ bins, using a hash function $h$: send element $x \in R$ to bin $h(x) \in [p]$.

Denote $L_u =$ number of elements in bin $u \in [p]$ (a random variable).

Example: $h(x) = [(x -'b') \mod 3] + 1$

\[
\begin{align*}
x & : \ a \ a \ a \ b \ c \ c \ d \ e \ e \ e \\
h(x) & : \ 3 \ 3 \ 3 \ 1 \ 2 \ 2 \ 3 \ 1 \ 1 \ 1 \\
\end{align*}
\]

$L_1 = 4$, $L_2 = 2$, $L_3 = 4$

Q: What is $E[L_u]$, for a fixed $u$?
Hash-Based Partition

Let $R$ be a bag ("multi-set") with $m$ elements.

Partition $R$ into $p$ bins, using a hash function $h$: send element $x \in R$ to bin $h(x) \in [p]$.

Denote $L_u =$ number of elements in bin $u \in [p]$ (a random variable).

Example: $h(x) = [(x' - b') \mod 3] + 1$

<table>
<thead>
<tr>
<th>$x$</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>e</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x)$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$L_1 = 4$, $L_2 = 2$, $L_3 = 4$

Q: What is $E[L_u]$, for a fixed $u$? A: $E[L_u] = m/p$
Hash-Based Partition

Let $R$ be a bag ("multi-set") with $m$ elements.

Partition $R$ into $p$ bins, using a hash function $h$: send element $x \in R$ to bin $h(x) \in [p]$.

Denote $L_u =$ number of elements in bin $u \in [p]$ (a random variable).

Example: $h(x) = [(x - b') \mod 3] + 1$

- $x: a \ a \ a \ b \ c \ c \ d \ e \ e \ e$
- $h(x): 3 \ 3 \ 3 \ 1 \ 2 \ 2 \ 3 \ 1 \ 1 \ 1$

$L_1 = 4, L_2 = 2, L_3 = 4$


- Q: What is $E[\max_u L_u]$?
Hash-Based Partition

Let $R$ be a bag ("multi-set") with $m$ elements.

Partition $R$ into $p$ bins, using a hash function $h$: send element $x \in R$ to bin $h(x) \in [p]$.

Denote $L_u =$ number of elements in bin $u \in [p]$ (a random variable).
Example: $h(x) = [(x - b') \mod 3] + 1$

\[
\begin{align*}
x & : a \ a \ a \ b \ c \ c \ d \ e \ e \ e \\
h(x) & : 3 \ 3 \ 3 \ 1 \ 2 \ 2 \ 3 \ 1 \ 1 \ 1
\end{align*}
\]

$L_1 = 4, L_2 = 2, L_3 = 4$

- Q: What is $E[\max_u L_u]$? A: May be as large as $m$. 
Hash-Based Partition

Let $d_x$ denote the number of occurrences of the value $x$ in $R$.

**Theorem**

Let $\alpha > 0$ be a constant such that $d_x \leq \frac{m}{\alpha \cdot p}$, for all $x$. Then, for all $\delta \geq 0$:

\[
\forall u \in [p] : \mathbb{P} \left( L_u > (1 + \delta) \frac{m}{p} \right) < \frac{1}{2^{\alpha \cdot \delta}} \quad \mathbb{P} \left( \max_u L_u > (1 + \delta) \frac{m}{p} \right) < \frac{p}{2^{\alpha \cdot \delta}}
\]

Exercise on the Problem Set: prove it (hint: use Bennett’s theorem).

Application: if $\alpha \delta \geq (1 + c) \log p$ for some constant $c > 0$, then

\[
\mathbb{P} \left( \max_u L_u > (1 + \delta) \frac{m}{p} \right) < \frac{1}{p^c}
\]
Summary on Hash-Based Partition

**Basic Partition**  Sent item $R(x)$ to bin $h(x)$.

If $R$ has *no skew* (degrees $\leq m/p$) then

Max-Size = $O(\text{Expected-Size}) = O(m/p)$, w.h.p. (hiding log $p$ factor)

**HyperCube Partition**  Send $R(x_1, \ldots, x_k)$ to bin $(h_1(x_1), \ldots, h_k(x_k))$

If $R$ has *no skew* then Max-Size = $O(\text{Expected-Size}) = O(m/p)$ w.h.p.

- Given shares $p_1, p_2, \ldots, p_k = p$, *no skew* means:

  $\forall S \subseteq [r]$, every value of $(x_i)_{i \in S}$ occurs $\leq m/\prod_{i \in S} p_i$ times:

  - $x_1 = a$ occurs $\leq m/p_1$ times
  - $x_2 = b$ occurs $\leq m/p_2$ times
  - $(x_1 = a, x_2 = b)$ occurs $\leq m/p_1 p_2$ times, etc.

- The hidden log $p$ factor becomes a poly-log factor.
Summary on Hash-Based Partition

**Basic Partition**  Sent item $R(x)$ to bin $h(x)$.
If $R$ has *no skew* (degrees $\leq m/p$) then
Max-Size $= O(\text{Expected-Size}) = O(m/p)$, w.h.p. (hiding log $p$ factor)

**HyperCube Partition**  Send $R(x_1, \ldots, x_k)$ to bin $(h_1(x_1), \ldots, h_k(x_k))$
If $R$ has *no skew* then Max-Size $= O(\text{Expected-Size}) = O(m/p)$ w.h.p.

- Given shares $p_1 p_2 \cdots p_k = p$, *no skew* means:
  $\forall S \subseteq [r], \text{every value of } (x_i)_{i \in S} \text{ occurs } \leq m/\prod_{i \in S} p_i \text{ times:}$
  - $x_1 = a$ occurs $\leq m/p_1$ times
  - $x_2 = b$ occurs $\leq m/p_2$ times
  - $(x_1 = a, x_2 = b)$ occurs $\leq m/p_1 p_2$ times, etc.

- The hidden log $p$ factor becomes a poly-log factor.
Summary on Hash-Based Partition

Basic Partition  Sent item $R(x)$ to bin $h(x)$.
If $R$ has no skew (degrees $\leq m/p$) then
Max-Size = $O(\text{Expected-Size}) = O(m/p)$, w.h.p. (hiding log $p$ factor)

HyperCube Partition  Send $R(x_1, \ldots, x_k)$ to bin $(h_1(x_1), \ldots, h_k(x_k))$
If $R$ has no skew then Max-Size = $O(\text{Expected-Size}) = O(m/p)$ w.h.p.

- Given shares $p_1p_2\cdots p_k = p$, no skew means:
  $\forall S \subseteq [r]$, every value of $(x_i)_{i \in S}$ occurs $\leq m/\prod_{i \in S} p_i$ times:
  $x_1 = a$  occurs $\leq m/p_1$ times
  $x_2 = b$  occurs $\leq m/p_2$ times
  $(x_1 = a, x_2 = b)$  occurs $\leq m/p_1p_2$ times, etc.

- The hidden log $p$ factor becomes a poly-log factor.
Summary on Hash-Based Partition

**Basic Partition**  Sent item $R(x)$ to bin $h(x)$.
If $R$ has no skew (degrees $\leq m/p$) then
Max-Size = $O(\text{Expected-Size}) = O(m/p)$, w.h.p. (hiding log $p$ factor)

**HyperCube Partition**  Send $R(x_1, \ldots, x_k)$ to bin $(h_1(x_1), \ldots, h_k(x_k))$
If $R$ has no skew then Max-Size = $O(\text{Expected-Size}) = O(m/p)$ w.h.p.

- Given shares $p_1p_2\cdots p_k = p$, no skew means:
  $\forall S \subseteq [r]$, every value of $(x_i)_{i \in S}$ occurs $\leq m/ \prod_{i \in S} p_i$ times:
  
  $x_1 = a$  occurs $\leq m/p_1$ times
  $x_2 = b$  occurs $\leq m/p_2$ times
  $(x_1 = a, x_2 = b)$  occurs $\leq m/p_1p_2$ times, etc.

- The hidden log $p$ factor becomes a poly-log factor.
Heavy Hitters

Definition

(Informal) A value (or tuple of values) is a heavy hitter if it occurs more often than its skew threshold.

Example:

Basic partition of $R(x, y)$ into $p$ bins: $x$ is heavy hitter if $d_x > m/p$

Hypercube partition of $R(x, y, z)$ into a cube $p_1p_2p_3$:

- $x$ is a heavy hitter if $d_x > m/p_1$.
- $y$ is a heavy hitter if $d_y > m/p_2$.
- A pair $(x, y)$ is a heavy hitter if $d_{xy} > m/p_1p_2$.
- A triple $(x, y, z)$ is never heavy. (Why?)
- etc

Fact

There are at most $O(p)$ heavy hitters. (Why?)
Heavy Hitters

Two ways to cope with heavy hitters:

**Unknown Heavy Hitters** Design the algorithm to be robust to heavy hitters.

**Known Heavy Hitters** Find all heavy hitters (only $O(p)$) and treat them specially. This is by far preferred in practice.
Unknown Heavy Hitters

Consider a join: \( Q(x, y, z) = R(x, y), S(y, z) \).

**Algorithm 1** Use shares \( p_x = 1, p_y = p, p_z = 1 \) (standard hash-join).
- If data has no skew: \( L = m/p \).
- If data is skewed: \( L = m \).  **Sensitive to skew**

**Algorithm 2** Use shares \( p_x = p_y = p_z = 1/3 \).
- If data has no skew: \( L = m/p^{2/3} \).
- If data is skewed: \( L = m/p^{1/3} \).  **Resilient to skew**

This observation generalizes: for any query, we can compute shares that minimize the load under the worst skew.
Known Heavy Hitters

Let $HH$ be the set of all heavy hitter values. $|HH| = O(p)$
For each relation $R_j(x_j)$, variables $z \subseteq x_j$, constants $v \in \text{Domain}^z$, let:

$$m_{j,z}[v] = \text{number of tuples in } R_j \text{ that have } z = v$$

In particular, $m_{j,\emptyset}[] = m_j$

**Definition**

Given a database instance $R_1, \ldots, R_\ell$, its statistics are the set of numbers:

$$\Sigma = \{m_{j,z}[v] \mid j = 1, \ell, z \subseteq x_j, v \subseteq HH^z\}$$

Note: $|\Sigma| = O(p)$, small enough that we can broadcast to all servers.

Open Problem design a $\Sigma$-optimal query evaluation algorithms (provably optimal for the set of statistics $\Sigma$).
Known Heavy Hitters

Let $HH$ be the set of all heavy hitter values. $|HH| = O(p)$

For each relation $R_j(x_j)$, variables $z \subseteq x_j$, constants $v \in \text{Domain}^z$, let:

$$m_{j,z}[v] = \text{number of tuples in } R_j \text{ that have } z = v$$

In particular, $m_{j,\emptyset}[] = m_j$

**Definition**

Given a database instance $R_1, \ldots, R_\ell$, its statistics are the set of numbers:

$$\Sigma = \{m_{j,z}[v] \mid j = 1, \ell, z \subseteq x_j, v \subseteq HH^z\}$$

Note: $|\Sigma| = O(p)$, small enough that we can broadcast to all servers.

Open Problem design a $\Sigma$-optimal query evaluation algorithms (provably optimal for the set of statistics $\Sigma$).
Known Heavy Hitters

Let $HH$ be the set of all heavy hitter values. \(|HH| = O(p)\)

For each relation $R_j(x_j)$, variables $z \subseteq x_j$, constants $v \in \text{Domain}^z$, let:

$$m_{j,z}[v] = \text{number of tuples in } R_j \text{ that have } z = v$$

In particular, $m_{j,\emptyset}()[()] = m_j$

**Definition**

Given a database instance $R_1, \ldots, R_\ell$, its statistics are the set of numbers:

$$\Sigma = \{ m_{j,z}[v] \mid j = 1, \ell, z \subseteq x_j, v \subseteq HH^z \}$$

Note: \(|\Sigma| = O(p)\), small enough that we can broadcast to all servers.

**Open Problem** design a $\Sigma$-optimal query evaluation algorithms (provably optimal for the set of statistics $\Sigma$).
Discussion

- A $\Sigma$-optimal algorithm represents the sweet spot between worst-case algorithm, and instance-optimal algorithms [Ngo’14].

- In practice, systems often compute on the fly some statistics in $\Sigma$ in order to avoid significant skew, e.g. skew-join in PigLatin.

- No $\Sigma$-optimal algorithm for arbitrary queries is known to date. [Beame’14] describe an algorithm that is optimal only within a poly-log $p$ factor (due to the algorithm, not to the hash function).

- $\Sigma$-optimal algorithms are known for a few special cases, in particular for a join [Beame’14]. We will discuss next.
**Σ-Optimal Join Algorithm**

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

\[ |R| = m_R, |S| = m_S. \]

∀ \( v \in \text{Domain} \) \[ m_R[v] = \text{degree of } v \text{ in } R \]

\[ m_S[v] = \text{degree of } v \text{ in } S \]

(Thus: \( \Sigma_v m_R[v] = m_R \) and \( \Sigma_v m_S[v] = m_S \))

Heavy hitters:

\[ HH_R = \{ v \mid m_R[v] \geq m_R/p \} \]

\[ HH_S = \{ v \mid m_S[v] \geq m_S/p \} \]

\[ HH = HH_R \cup HH_S \]

Statistics:

\[ \Sigma = \{ m_R, m_S \} \cup \{ m_R[v] \mid v \in HH_R \} \cup \{ m_S[v] \mid v \in HH_S \} \]
**Σ-Optimal Join Algorithm**

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

\[ |R| = m_R, |S| = m_S. \]

\[ \forall v \in \text{Domain} \]

\[ m_R[v] = \text{degree of } v \text{ in } R \]

\[ m_S[v] = \text{degree of } v \text{ in } S \]

(Thus: \( \sum_v m_R[v] = m_R \) and \( \sum_v m_S[v] = m_S \))

Heavy hitters:

\[ HH_R = \{ v \mid m_R[v] \geq m_R/p \} \]

\[ HH_S = \{ v \mid m_S[v] \geq m_S/p \} \]

\[ HH = HH_R \cup HH_S \]

Statistics:

\[ \Sigma = \{ m_R, m_S \} \cup \{ m_R[v] \mid v \in HH_R \} \cup \{ m_S[v] \mid v \in HH_S \} \]
Σ-Optimal Join Algorithm

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

\[ |R| = m_R, |S| = m_S. \]

∀ \( v \in \text{Domain} \)

\[ m_R[v] = \text{degree of } v \text{ in } R \]

\[ m_S[v] = \text{degree of } v \text{ in } S \]

(Thus: \( \sum_v m_R[v] = m_R \) and \( \sum_v m_S[v] = m_S \))

Heavy hitters:

\[ HH_R = \{ v \mid m_R[v] \geq m_R/p \} \]

\[ HH_S = \{ v \mid m_S[v] \geq m_S/p \} \]

\[ HH = HH_R \cup HH_S \]

Statistics:

\[ \Sigma = \{ m_R, m_S \} \cup \{ m_R[v] \mid v \in HH_R \} \cup \{ m_S[v] \mid v \in HH_S \} \]
**Σ-Optimal Join Algorithm**

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

\[ \forall v \in \text{Domain} \quad m_R[v] = \text{degree of } v \text{ in } R \]
\[ m_S[v] = \text{degree of } v \text{ in } S \]

(Thus: \( \sum_v m_R[v] = m_R \) and \( \sum_v m_S[v] = m_S \))

**Heavy hitters:**

\[ HH_R = \{ v \mid m_R[v] \geq m_R/p \} \]
\[ HH_S = \{ v \mid m_S[v] \geq m_S/p \} \]
\[ HH = HH_R \cup HH_S \]

**Statistics:**

\[ \Sigma = \{ m_R, m_S \} \cup \{ m_R[v] \mid v \in HH_R \} \cup \{ m_S[v] \mid v \in HH_S \} \]
Cartesian Product Revisited

A heavy hitter transforms the join into a cartesian product, let's revisit it

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

The load is: \[ L = \max_u \left( \frac{m_1^{u_1} m_2^{u_2}}{p} \right)^{1/(u_1+u_2)} \]

<table>
<thead>
<tr>
<th>u</th>
<th>L(u)</th>
<th>Shares ( p_x, p_z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1)</td>
<td>((m_R m_S / p)^{1/2})</td>
<td>(\sqrt{p m_R / m_S}, \sqrt{p m_S / m_R})</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>(m_R / p)</td>
<td>(p, 1)</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>(m_S / p)</td>
<td>(1, \sqrt{p} )</td>
</tr>
</tbody>
</table>

\[ L = \max((m_R m_S / p)^{1/2}, m_R / p, m_S / p) \]
$\Sigma$-Optimal Join Algorithm – Intuition

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

- $\forall v \in HH$ a cartesian product: $Q_v(x, z) = R_v(x), S_v(z)$, where $R_v(x) = R(x, v), S_v(z) = S(v, y)$.

- $Q_v = \text{residual query}$; $|R_v| = m_R[v], |S_v| = m_S[v]$.

- Allocate $p_v \leq p$ servers to compute $Q_v$.
  Load $L_v = (m_R[v] \cdot m_S[v]/p_v)^{1/2}$ (assuming $> m_R[v]/p, m_S[v]/p$)

- Determine the number of servers $p_v$ such that $\sum_{v \in HH} p_v = p$, and $\max_v L_v$ is minimized.

- Optimal solution: $p_v = p \frac{m_R[v]m_S[v]}{\sum_{w \in HH} m_R[w]m_S[w]}$

- Optimal load $L = \left(\frac{\sum_{w \in HH} m_R[w]m_S[w]}{p}\right)^{1/2} = L_v$, for all $v \in HH$. 
Σ-Optimal Join Algorithm – Intuition

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

- \( \forall v \in HH \) a cartesian product: \( Q_v(x, z) = R_v(x), S_v(z) \), where \( R_v(x) = R(x, v), S_v(z) = S(v, y) \).

- \( Q_v = \text{residual query}; |R_v| = m_R[v], |S_v| = m_S[v] \).

- Allocate \( p_v \leq p \) servers to compute \( Q_v \).
  Load \( L_v = (m_R[v] \cdot m_S[v]/p_v)^{1/2} \) (assuming \( > m_R[v]/p, m_S[v]/p \))

- Determine the number of servers \( p_v \) such that \( \sum_{v \in HH} p_v = p \), and \( \max_v L_v \) is minimized.

- Optimal solution: \( p_v = p \frac{m_R[v]m_S[v]}{\sum_{w \in HH} m_R[w]m_S[w]} \)

- Optimal load \( L = \left( \frac{\sum_{w \in HH} m_R[w]m_S[w]}{p} \right)^{1/2} = L_v \), for all \( v \in HH \).
Σ-Optimal Join Algorithm – Intuition

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

- \( \forall v \in HH \) a cartesian product: \( Q_v(x, z) = R_v(x), S_v(z) \), where \( R_v(x) = R(x, v), S_v(z) = S(v, y) \).
- \( Q_v = \text{residual query}; |R_v| = m_R[v], |S_v| = m_S[v] \).
- Allocate \( p_v \leq p \) servers to compute \( Q_v \).
  Load \( L_v = (m_R[v] \cdot m_S[v]/p_v)^{1/2} \) (assuming \( > m_R[v]/p, m_S[v]/p \))
  - Determine the number of servers \( p_v \) such that \( \sum_{v \in HH} p_v = p \), and \( \max_v L_v \) is minimized.
- Optimal solution: \( p_v = p \frac{m_R[v]m_S[v]}{\sum_{w \in HH} m_R[w]m_S[w]} \)
- Optimal load \( L = \left( \frac{\sum_{w \in HH} m_R[w]m_S[w]}{p} \right)^{1/2} = L_v \), for all \( v \in HH \).
\( \Sigma \)-Optimal Join Algorithm – Intuition

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

- \( \forall v \in HH \) a cartesian product: \( Q_v(x, z) = R_v(x), S_v(z) \), where \( R_v(x) = R(x, v), S_v(z) = S(v, y) \).

- \( Q_v = \text{residual query}; |R_v| = m_R[v], |S_v| = m_S[v] \).

- Allocate \( p_v \leq p \) servers to compute \( Q_v \).
  Load \( L_v = (m_R[v] \cdot m_S[v])/p_v)^{1/2} \) (assuming > \( m_R[v]/p, m_S[v]/p \))

- Determine the number of servers \( p_v \) such that \( \sum_{v \in HH} p_v = p \), and \( \max_v L_v \) is minimized.

  - Optimal solution: \( p_v = p \frac{m_R[v] m_S[v]}{\sum_{w \in HH} m_R[w] m_S[w]} \)

  - Optimal load \( L = \left( \frac{\sum_{w \in HH} m_R[w] m_S[w]}{p} \right)^{1/2} = L_v \), for all \( v \in HH \).
Σ-Optimal Join Algorithm – Intuition

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

- \( \forall v \in HH \) a cartesian product: \( Q_v(x, z) = R_v(x), S_v(z) \), where \( R_v(x) = R(x, v), S_v(z) = S(v, y) \).
- \( Q_v = \text{residual query}; |R_v| = m_R[v], |S_v| = m_S[v] \).
- Allocate \( p_v \leq p \) servers to compute \( Q_v \).
  Load \( L_v = (m_R[v] \cdot m_S[v]/p_v)^{1/2} \) (assuming \( > m_R[v]/p, m_S[v]/p \))
- Determine the number of servers \( p_v \) such that \( \sum_{v \in HH} p_v = p \), and \( \max_v L_v \) is minimized.
- Optimal solution:
  \[ p_v = p \frac{m_R[v] m_S[v]}{\sum_{w \in HH} m_R[w] m_S[w]} \]
- Optimal load \( L = \left( \frac{\sum_{w \in HH} m_R[w] m_S[w]}{p} \right)^{1/2} = L_v \), for all \( v \in HH \).
\(\Sigma\)-Optimal Join Algorithm – Intuition

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

- \(\forall v \in HH\) a cartesian product: \(Q_v(x, z) = R_v(x), S_v(z)\), where \(R_v(x) = R(x, v), S_v(z) = S(v, y)\).

- \(Q_v = \text{residual query}; |R_v| = m_{R[v]}, |S_v| = m_{S[v]}\).

- Allocate \(p_v \leq p\) servers to compute \(Q_v\).

  Load \(L_v = (m_{R[v]} \cdot m_{S[v]}/p_v)^{1/2}\) (assuming \(> m_{R[v]}/p, m_{S[v]}/p\))

- Determine the number of servers \(p_v\) such that \(\sum_{v\in HH} p_v = p\), and \(\max_v L_v\) is minimized.

- Optimal solution: \(p_v = p \frac{m_{R[v]} m_{S[v]}}{\sum_{w\in HH} m_{R[w]} m_{S[w]}}\)

- Optimal load \(L = \left(\frac{\sum_{w\in HH} m_{R[w]} m_{S[w]}}{p}\right)^{1/2} = L_v\), for all \(v \in HH\).
Algorithm HyperSkew

Assume $HH_R = HH_S = HH$

**Light Hitters** Run HyperCube on the light hitters.

Load: $\max(m_R/p, m_S/p)$

**Heavy Hitters** In parallel, for each $v \in HH$:

Compute $Q_v$ using HyperCube on $p_v = p \frac{m_R[v] m_S[v]}{\sum_{w \in HH} m_R[w] m_S[w]}$ servers.

Load: $L = \left( \frac{\sum_{w \in HH} m_R[w] m_S[w]}{p} \right)^{1/2}$

Exercise: generalize to $HH_R \neq HH_S$ (Hint: set $m_R[v] = 1$ for $v \in HH_S - HH_R$ etc).

The total load is:

$$L_{\text{lower}} = \max(m_R/p, m_S/p, \left( \frac{\sum_{w \in HH} m_R[w] m_S[w]}{p} \right)^{1/2})$$
One Sided Heavy Hitters

The contribution of one-sided heavy hitters is negligible:

**Lemma**

\[
\left( \frac{\sum_{w \in HH_R - HH_S} m_R[w] m_s[w]}{p} \right)^{1/2} \leq \max(m_R, m_S)/p
\]

**Proof.**

\[
\sum_{w \in HH_R - HH_S} m_R[w] m_s[w] \leq \left( \sum_{w \in HH_R - HH_S} m_R[w] \right) \frac{m_s}{p} \leq \frac{m_R m_s}{p}
\]

Thus, the load due to one sided heavy hitters will not exceed the load due to light hitters, and it’s OK to approximate the missing degree with 1.
Discussion

- Skew may affect significantly the communication cost for joins: the speedup decreases from $1/p$ to $1/p^{1/2}$

- Adapting the algorithm to the statistics $\Sigma$ is also important: if all heavy hitters are one sided, then the extra cost can be avoided completely.
Multiple Rounds

- Basic idea is very simple: generate a query plan for \( Q \), then compute the plan bottom up, each level is one round.
- Goal: reduce the load by having multiple rounds.
- Challenge (major!): intermediate results may be much bigger, and are hard to estimate.

Two approaches:

- In [Beame’13] each operator in the query plan is a conjunctive query (rather than just a single join), and is evaluated using 1-round HyperCube. No control over the intermediate results.
- In [Afrati’14] the GYM algorithm computes conjunctive queries only at the leaves s.t. the residual query is acyclic, then use Yannakakis’ semi-join reduction to control the size of the intermediate results.
Grand Summary

- Big Data Analytics needs to run complex queries, on big data.

- Traditional query processing: one join at a time. Challenge: large and unpredictable intermediate results.

- Novel worst-case optimal sequential algorithm: runtime = AGM bound.

- Novel parallel algorithm: communication cost = provably optimal.

- Many open problems (next)
Open Problem 1: AGM Bounds for Given Statistics

Generalize the AGM bound to databases with known statistics $\Sigma$.

Simpler problems:

Generalize the AGM bound for databases with bounded degrees. E.g. $Q = R(x, y), S(y, z)$, normal AGM upper bound is $m^2$, if all degrees $\leq d$, upper bound is $dm$.

Generalize the AGM bound to functional dependencies. This immediately proves the previous item (for details, send me email)
Open Problem 2: $\Sigma$-Optimal Sequential Algorithm

Design a sequential algorithm that is worst-case optimal for instances satisfying given statistics $\Sigma$.

E.g. $Q = R(x, y), S(y, z), T(z, x)$. Suppose $\forall y$, degree of $y$ in $S$ is $\leq 5$. Then compute $Q$ in time $O(m)$. What if we know one HH $y_0$ with degree $m/2$?
Open Problem 3: Lower Bounds for Multi-Round Parallel Algorithm

Prove lower bounds for 2 or more rounds, assuming servers can send messages encoding arbitrary information.

Example: prove that $R(x, y), S(y, z), T(z, u), K(u, v), L(v, w)$ cannot be computed in 2 rounds, with load $m/p$.

Note: [Beame’13] proved such lower bounds, but for a weaker model, when message consists of tuples, not of arbitrary bits.
Open Problem 4: Optimal Multi-round Algorithms

Find the exact tradeoff between the load/round $L$ and the number of rounds $r$ for a general query.

Note: emphasis here is on algorithm, not lower bounds. For lower bounds it’s OK to give the proof for a weaker models (as in [Beame’13]).
Open Problem 5: \( \Sigma \)-Optimal Parallel Algorithms

Generalize HyperSkew from joins to an arbitrary queries. Seems straightforward how to generalize it, but it's open whether this is optimal.

Conjectured tight bound:

\[
L_{\text{lower}} = \max_{u,z} \left( \frac{\sum_{v \in \text{Domain}^z} m_{1,z}^u[v] \cdots m_{\ell,z}^u[v]}{p} \right)^{1/\sum_j u_j}
\]

where \( z \subseteq \{x_1, \ldots, x_k\} \), and \( u \) is fractional edge packing of the residual query \( q_z \) that covers all variables in \( z \).

This formula is known to be a lower bound [Beame’14], but not known if tight.
Final Remark

- Please solve the 7 problems on the Problem Set. Some are quite easy, some more challenging.

- If you work on any of the open problems and make progress, please let me know.
Thank you!