Multi-join Query Evaluation on Big Data
Lecture 2

Dan Suciu

March, 2015

Multi-join Query Evaluation – Outline

Part 1 Optimal Sequential Algorithms. Thursday 14:15-15:45

Part 2 Lower bounds for Parallel Algorithms. Friday 14:15-15:45

Part 3 Optimal Parallel Algorithms. Saturday 9-10:30

Part 3 Data Skew. Saturday 11-12:30

Brief Review of the AGM Bound

$$Q(\mathbf{x}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$$

Equal Cardinalities: $|R_1| = \ldots |R_\ell| = m$

$|Q| \leq m^{\rho^*}$, where ρ^* = fractional edge covering number of Q.

General Case: $R_1 = m_1, \ldots, R_\ell = m_\ell$

$|Q| \leq \min_{\mathbf{u}} m_1^{u_1} \cdots m_\ell^{u_\ell}$

Outline for Lecture 2

- Background: Parallel Databases
- The MPC Model
- Algorithm for Triangles
- Lower Bound on the Communication Cost
- Lower Bound for General Queries
- Summary

Parallel Query Processing: Overview

Parallel Databases: Queries and updates

- GAMMA machine (80's), today in all commercial DBMS.
- Typically x10's nodes.
- FO in AC^0; "SQL embarrassingly parallel"

Parallel Data Analytics: Queries only – this course

- MapReduce, Hadoop, PigLatin, Dremel, Scope, Spark.
- Typically x100's or x1000's nodes.
- Data reshuffling / communication is new bottleneck.

Distributed State: Updates (transactions) – will not discuss

- Replicate objects (3-5 times). Central problem: consistency.
- E.g. Google, Amazon, Yahoo, Microsoft, Ebay.
- Eventual consistency (NoSQL, Dynamo, BigTable, Peanuts).
- Strong consistency: Paxos, Spanner.

Parallel Query Processing: Overview

Parallel Databases: Queries and updates

- GAMMA machine (80's), today in all commercial DBMS.
- Typically x10's nodes.
- FO in AC^0; "SQL embarrassingly parallel"

Parallel Data Analytics: Queries only – this course

- MapReduce, Hadoop, PigLatin, Dremel, Scope, Spark.
- Typically x100's or x1000's nodes.
- Data reshuffling / communication is new bottleneck.

Distributed State: Updates (transactions) – will not discuss

- Replicate objects (3-5 times). Central problem: consistency.
- E.g. Google, Amazon, Yahoo, Microsoft, Ebay.
- Eventual consistency (NoSQL, Dynamo, BigTable, Peanuts).
- Strong consistency: Paxos, Spanner.

Parallel Query Processing: Overview

Parallel Databases: Queries and updates

- GAMMA machine (80's), today in all commercial DBMS.
- Typically x10's nodes.
- FO in AC^0; "SQL embarrassingly parallel"

Parallel Data Analytics: Queries only – this course

- MapReduce, Hadoop, PigLatin, Dremel, Scope, Spark.
- Typically x100's or x1000's nodes.
- Data reshuffling / communication is new bottleneck.

Distributed State: Updates (transactions) – will not discuss

- Replicate objects (3-5 times). Central problem: consistency.
- E.g. Google, Amazon, Yahoo, Microsoft, Ebay.
- Eventual consistency (NoSQL, Dynamo, BigTable, Peanuts).
- Strong consistency: Paxos, Spanner.

Key Concepts

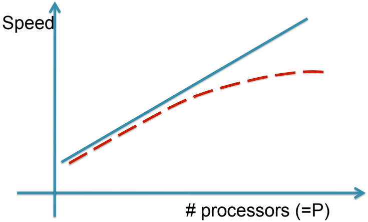
Input data of size m is partitioned on p servers, connected by a network.

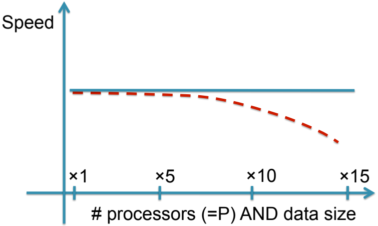
Query processing involves local computation and communication.

Performance parameters [Gray&Dewitt'92]

Speedup How does performance change when p increases?
Ideal: linear speedup.

Scaleup How does performance chance when p, m increase at same rate?
Ideal: constant scaleup.





Data Partition

Given a database of size m, partition it on p servers.

Balanced partition Each server holds $\approx m/p$ data.

Skewed partition Some server holds $\gg m/p$ data.

Usually, the input data is already partitioned, but we need to re-partition for a particular problem. *Data reshuffling*.

Example 1: Hash-Partitioned Join

Compute $Q(x, y, z) = R(x, y) \bowtie S(y, z)$, where $|R| = m_1$, $|S| = m_2$

Input data The input relations R, S are partitioned on the servers.

Data reshuffling Given hash function h : Domain $\rightarrow [p]$:
Send every tuple $R(x, y)$ to server $h(y)$.
Send every tuple $S(y, z)$ to server $h(y)$.

Computation In parallel, each server i computes the join
$R_i(x, y) \bowtie S_i(y, z)$ of its local fragments R_i, S_i.

If y is a key, then the load/server is $m_1/p + m_2/p$ (why?) *Linear Speedup*

Otherwise, we may have skew (why?) What is the worst speedup?

Understanding Skew

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

Claim 1 For each server u, its expected load is $\mathbf{E}[L_u] = m/p$. Why?

Claim 2 We say that R is *skewed* if some value y occurs more than m/p times in R. Then some server has $L_u > m/p$.

Claim 3 If R is not skewed, the $\max_u L_u$ is $O(m/p)$ with high probability. (Details in lecture 4.)

Take-away: we assume *no skew*, then:

$$\text{Max-load} = O(\text{Expected load})$$

Understanding Skew

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

Claim 1 For each server u, its expected load is $\mathbf{E}[L_u] = m/p$. Why?

Claim 2 We say that R is *skewed* if some value y occurs more than m/p times in R. Then some server has $L_u > m/p$.

Claim 3 If R is not skewed, the $\max_u L_u$ is $O(m/p)$ with high probability. (Details in lecture 4.)

Take-away: we assume *no skew*, then:

$$\text{Max-load} = O(\text{Expected load})$$

Understanding Skew

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

Claim 1 For each server u, its expected load is $\mathbf{E}[L_u] = m/p$. Why?

Claim 2 We say that R is *skewed* if some value y occurs more than m/p times in R. Then some server has $L_u > m/p$.

Claim 3 If R is not skewed, the $\max_u L_u$ is $O(m/p)$ with high probability. (Details in lecture 4.)

Take-away: we assume *no skew*, then:

$$\text{Max-load} = O(\text{Expected load})$$

Understanding Skew

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

Claim 1 For each server u, its expected load is $\mathbf{E}[L_u] = m/p$. Why?

Claim 2 We say that R is *skewed* if some value y occurs more than m/p times in R. Then some server has $L_u > m/p$.

Claim 3 If R is not skewed, the $\max_u L_u$ is $O(m/p)$ with high probability. (Details in lecture 4.)

Take-away: we assume *no skew*, then:

$$\text{Max-load} = O(\text{Expected load})$$

Example 2: Broadcast Join

Compute $Q(x, y, z) = R(x, y) \bowtie S(y, z)$, where $|R| = m_1 \gg |S| = m_2$

Input data The input relations R, S are partitioned on the servers.

Broadcast Send every tuple $S(y, z)$ to every server

Computation In parallel, each server u computes the join $R_u(x, y) \bowtie S(y, z)$ of its local fragment R_u with S.

If $m_2 \leq m_1/p$ then the Broadcast Join is very effective. Used a lot in practice.

Massively Parallel Communication Model (MPC)

[Beame'13] The MPC model is the following:

p servers are connected by a network. Servers are infinitely powerful.

Input Data of size m is initially uniformly partitioned.

Computation = several rounds.

One round = local computation plus global communication.

Massively Parallel Communication Model (MPC)

[Beame'13] The MPC model is the following:

p servers are connected by a network. Servers are infinitely powerful.

Input Data of size m is initially uniformly partitioned.

Computation = several rounds.

One round = local computation plus global communication.

The only cost is the communication.

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

L_u = maximum amount of data received by server u during any round
$L = \max_u L_u$

Massively Parallel Communication Model (MPC)

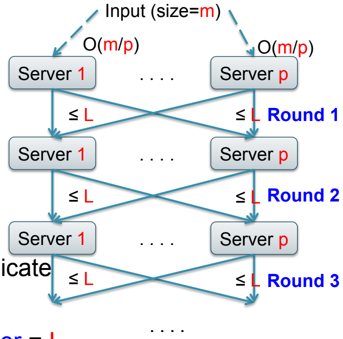
Number of servers = p

Input data = size m

Algorithm = Several rounds

One round = Compute & communicate

Max communication load per server = L



Load/Rounds Tradeoff in the MPC Model

Naive 1-Round Send entire data to server 1, compute locally. $L = m$

Naive p-Rounds At each round, send a m/p-fragment of the data to server 1, then compute locally. $L = m/p$.

Ideal Algorithms 1-Round, load $L = m/p$ (but rarely possible)

Real Algorithms $O(1)$ rounds, and $L = O(m/p^{1-\varepsilon})$, for $0 \leq \varepsilon < 1$.

Load/Rounds Tradeoff in the MPC Model

Naive 1-Round Send entire data to server 1, compute locally. $L = m$

Naive p-Rounds At each round, send a m/p-fragment of the data to server 1, then compute locally. $L = m/p$.

Ideal Algorithms 1-Round, load $L = m/p$ (but rarely possible)

Real Algorithms $O(1)$ rounds, and $L = O(m/p^{1-\varepsilon})$, for $0 \leq \varepsilon < 1$.

Load/Rounds Tradeoff in the MPC Model

Naive 1-Round Send entire data to server 1, compute locally. $L = m$

Naive p-Rounds At each round, send a m/p-fragment of the data to server 1, then compute locally. $L = m/p$.

Ideal Algorithms 1-Round, load $L = m/p$ (but rarely possible)

Real Algorithms $O(1)$ rounds, and $L = O(m/p^{1-\varepsilon})$, for $0 \leq \varepsilon < 1$.

Load/Rounds Tradeoff in the MPC Model

Naive 1-Round Send entire data to server 1, compute locally. $L = m$

Naive p-Rounds At each round, send a m/p-fragment of the data to server 1, then compute locally. $L = m/p$.

Ideal Algorithms 1-Round, load $L = m/p$ (but rarely possible)

Real Algorithms $O(1)$ rounds, and $L = O(m/p^{1-\varepsilon})$, for $0 \le \varepsilon < 1$.

Examples on the MPC Model

Example: Join $Q(x, y, z) = R(x, y), S(y, z)$.

Round 1 (Hash-partitioned join) Send tuple $R(x, y)$ to server $h(y)$, send $S(y, z)$ to server $h(y)$, compute the join locally.

Load: $O(m/p)$ (assuming no skew).

Example: Triangles $Q(x, y, z) = R(x, y), S(y, z), T(z, x)$

Round 1 hash-partitioned join: $\text{Aux}(x, y, z) = R(x, y), S(y, z)$

Round 2 hash-partitioned join: $Q(x, y, z) = \text{Aux}(x, y, z), T(z, x)$

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

Can we compute triangles with a smaller load?

Examples on the MPC Model

Example: Join $Q(x, y, z) = R(x, y), S(y, z)$.

Round 1 (Hash-partitioned join) Send tuple $R(x, y)$ to server $h(y)$, send $S(y, z)$ to server $h(y)$, compute the join locally.

Load: $O(m/p)$ (assuming no skew).

Example: Triangles $Q(x, y, z) = R(x, y), S(y, z), T(z, x)$

Round 1 hash-partitioned join: $\text{Aux}(x, y, z) = R(x, y), S(y, z)$

Round 2 hash-partitioned join: $Q(x, y, z) = \text{Aux}(x, y, z), T(z, x)$

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

Can we compute triangles with a smaller load?

A Simple Lower Bound for the MPC Model

Let Q be a query, with $|R_1| = \cdots = |R_\ell| = m$.

Fact

For any algorithm for Q with r rounds and load L, it holds: $r \cdot L \geq m/p^{1/\rho^*}$.

A Simple Lower Bound for the MPC Model

Let Q be a query, with $|R_1| = \cdots = |R_\ell| = m$.

Fact

For any algorithm for Q with r rounds and load L, it holds: $r \cdot L \geq m/p^{1/\rho^*}$.

Proof

- Construct an instance where $|Q| = m^{\rho^*}$.
- One server: receives $\leq r \cdot L$ tuples from R_j, for all j, hence finds $\leq (r \cdot L)^{\rho^*}$ answers.
- All p servers find $p(r \cdot L)^{\rho^*}$ answers.
- It follows: $r \cdot L \geq m/p^{1/\rho^*}$. $\square$

A Simple Lower Bound for the MPC Model

Let Q be a query, with $|R_1| = \cdots = |R_\ell| = m$.

Fact

For any algorithm for Q with r rounds and load L, it holds: $r \cdot L \geq m/p^{1/\rho^*}$.

Proof

- Construct an instance where $|Q| = m^{\rho^*}$.
- One server: receives $\leq r \cdot L$ tuples from R_j, for all j, hence finds $\leq (r \cdot L)^{\rho^*}$ answers.
- All p servers find $p(r \cdot L)^{\rho^*}$ answers.
- It follows: $r \cdot L \geq m/p^{1/\rho^*}$. $\square$

Question

For $Q = R(x, y), S(y, z)$ it should be $r \cdot L \geq m/p^{1/2}$, but we computed a join with load $L = m/p$. Contradiction?

A Simple Lower Bound for the MPC Model

Let Q be a query, with $|R_1| = \cdots = |R_\ell| = m$.

Fact

For any algorithm for Q with r rounds and load L, it holds: $r \cdot L \geq m/p^{1/\rho^*}$.

Proof

- Construct an instance where $|Q| = m^{\rho^*}$.
- One server: receives $\leq r \cdot L$ tuples from R_j, for all j, hence finds $\leq (r \cdot L)^{\rho^*}$ answers.
- All p servers find $p(r \cdot L)^{\rho^*}$ answers.
- It follows: $r \cdot L \geq m/p^{1/\rho^*}$. $\square$

Question

For $Q = R(x, y), S(y, z)$ it should be $r \cdot L \geq m/p^{1/2}$, but we computed a join with load $L = m/p$. Contradiction?

Answer: the algorithm is for skew-free data, the bound for skewed data.

One-Round Algorithm for Triangles

[Afrati&Ullman'10,Suri&Vassilvitski'11]

$Q(x, y, z) = R(x, y), S(y, z), T(z, x)$
Place the p servers in a cube: $[p] \equiv [p^{1/3}] \times [p^{1/3}] \times [p^{1/3}]$.

Round 1 In parallel, each server does the following:
– Send $R(x, y)$ to all servers $(h_1(x), h_2(y), *)$
– Send $S(y, z)$ to all servers $(*, h_2(y), h_3(z))$
– Send $T(z, x)$ to all servers $(h_1(x), *, h_3(z))$
Then compute Q locally

Theorem (Beame'14)

(1) If h_1, h_2, h_3 are independent hash functions, then the expected load at some server u is $\mathbf{E}[L_u] = \frac{m_1+m_2+m_3}{p^{2/3}} \stackrel{def}{=} L$. [Why do we need independence?]
(2) If the data has no skew, then maximum load is $O(L)$ w.h.p.

One-Round Algorithm for Triangles

[Afrati&Ullman'10,Suri&Vassilvitski'11]

$Q(x, y, z) = R(x, y), S(y, z), T(z, x)$
Place the p servers in a cube: $[p] \equiv [p^{1/3}] \times [p^{1/3}] \times [p^{1/3}]$.

Round 1 In parallel, each server does the following:
– Send $R(x, y)$ to all servers $(h_1(x), h_2(y), *)$
– Send $S(y, z)$ to all servers $(*, h_2(y), h_3(z))$
– Send $T(z, x)$ to all servers $(h_1(x), *, h_3(z))$
Then compute Q locally

Theorem (Beame'14)

(1) If h_1, h_2, h_3 are independent hash functions, then the expected load at some server u is $\mathbf{E}[L_u] = \frac{m_1+m_2+m_3}{p^{2/3}} \stackrel{def}{=} L$. [Why do we need independence?]
(2) If the data has no skew, then maximum load is $O(L)$ w.h.p.

One-Round Algorithm for Triangles

[Afrati&Ullman'10,Suri&Vassilvitski'11]

$Q(x,y,z) = R(x,y), S(y,z), T(z,x)$
Place the p servers in a cube: $[p] \equiv [p^{1/3}] \times [p^{1/3}] \times [p^{1/3}]$.

Round 1 In parallel, each server does the following:
– Send $R(x,y)$ to all servers $(h_1(x), h_2(y), *)$
– Send $S(y,z)$ to all servers $(*, h_2(y), h_3(z))$
– Send $T(z,x)$ to all servers $(h_1(x), *, h_3(z))$
Then compute Q locally

Theorem (Beame'14)

(1) If h_1, h_2, h_3 are independent hash functions, then the expected load at some server u is $\mathbf{E}[L_u] = \frac{m_1+m_2+m_3}{p^{2/3}} \stackrel{def}{=} L$. [Why do we need independence?]
(2) If the data has no skew, then maximum load is $O(L)$ w.h.p.

Discussion

- We will call the algorithm *HyperCube*, following [Beame'13].
- Each tuple $R(x, y)$ is replicated $p^{1/3}$ times. Hence, load $L = m/p^{2/3}$
- Partitioning $R(x, y) \to (h_1(x), h_2(y), *)$ more tolerant to skew: Each value x is sent to $p^{1/3}$ buckets, each y is sent to $p^{1/3}$ buckets. Can tolerate degrees up to $\leq m/p^{1/3}$ (better than m/p).
- Notice that the algorithm only shuffles the data: each server still has to compute the query locally. Important to use worst-case algorithm.
- Non-linear speedup, because the load is $m/p^{2/3}$. Can we we compute triangles with a load of m/p?

Lower Bound for Triangle Queries

$Q(x, y, z) = R(x, y), S(y, z), T(z, x)$
Assume $|R| + |S| + |T| = m$. Denote $L_{\text{lower}} = m/p^{2/3}$.

Theorem

Any 1-round algorithm that computes Q must have load $L \geq L_{lower}$, even on database instances without skew.

Hence, HyperCube is optimal for computing triangles on permutations.

Next, we will prove the theorem.

Lower Bound for Triangle Queries

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

We assume R, S, T are permutations over a domain of size n.

Example $n = 4$:

R

x	y
1	3
2	1
3	4
4	2

$\bowtie$ S

y	z
1	4
2	2
3	1
4	3

$\bowtie$ T

z	x
1	1
2	2
3	3
4	4

$=$ Q

x	y	z
1	3	1
3	4	3

Question: what is $\mathbf{E}[|Q|]$, over random permutations R, S, T?

Theorem (Beame'13)

Denote $L_{lower} = 3n/p^{2/3}$. Let A be an algorithm for Q, with load $L < L_{lower}$. Then, the expected number of triangles returned by A is

$$\mathbf{E}[|A|] \leq \left(\frac{L}{L_{lower}}\right)^{3/2} \mathbf{E}[|Q|]$$

Lower Bound for Triangle Queries

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

We assume R, S, T are permutations over a domain of size n.

Example $n = 4$:

R

x	y
1	3
2	1
3	4
4	2

$\bowtie$

S

y	z
1	4
2	2
3	1
4	3

$\bowtie$

T

z	x
1	1
2	2
3	3
4	4

$=$

Q

x	y	z
1	3	1
3	4	3

Question: what is $\mathbf{E}[|Q|]$, over random permutations R, S, T?

Theorem (Beame'13)

Denote $L_{lower} = 3n/p^{2/3}$. Let A be an algorithm for Q, with load $L < L_{lower}$. Then, the expected number of triangles returned by A is

$$\mathbf{E}[|A|] \leq \left(\frac{L}{L_{lower}}\right)^{3/2} \mathbf{E}[|Q|]$$

Lower Bound for Triangle Queries

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

We assume R, S, T are permutations over a domain of size n.

Example $n = 4$:

R

x	y
1	3
2	1
3	4
4	2

$\bowtie$ S

y	z
1	4
2	2
3	1
4	3

$\bowtie$ T

z	x
1	1
2	2
3	3
4	4

$=$ Q

x	y	z
1	3	1
3	4	3

Question: what is $\mathbf{E}[|Q|]$, over random permutations R, S, T?

Theorem (Beame'13)

Denote $L_{lower} = 3n/p^{2/3}$. Let A be an algorithm for Q, with load $L < L_{lower}$. Then, the expected number of triangles returned by A is

$$\mathbf{E}[|A|] \leq \left(\frac{L}{L_{lower}}\right)^{3/2} \mathbf{E}[|Q|]$$

Discussion: Inputs, Messages, Bits

- Initially, the relations R, S, T are on disjoint servers. This is w.l.o.g. Stronger: assume that R is on server 1, S on server 2, T on server 3. Any server u receives three messages:
  – $\mathtt{msg}_1$ about R
  – $\mathtt{msg}_2$ about S
  – $\mathtt{msg}_3$ about T
  $|\mathtt{msg}_1| + |\mathtt{msg}_2| + |\mathtt{msg}_3| \leq L$ bits.
- Messages may encode arbitrary information.
  E.g. bit 1 says "R is even"; bit 2 says "R has a 17-cycle"; etc.
- No lower bound is possible for a *fixed* input R, S, T: an algorithm may simply check for that input and encode using $L = O(1)$ bits. Instead, the lower bound is a statement about *all* permutations.

Notation: Bits v.s. Tuples

- Size of db in #tuples $m = |R| + |S| + |T|$.
  Size of db in #bits $M = |R| \log n + |S| \log n + |T| \log n = m \log n$
  where n = size of the domain.
- If R is a random permutation, then size$(R) = \log(n!) \lesssim n \log n$ bits.
- $L_{\text{lower}} = m/p^{2/3}$ tuples becomes $L_{\text{lower}} = 3 \log(n!)/p^{2/3}$ bits.
- Will prove the following, where L, L_{lower} are expressed in bits:

$$\boxed{\mathbf{E}[|A|] \leq \left(\frac{L}{L_{\text{lower}}}\right)^{3/2} \mathbf{E}[|Q|]}$$

Proof – Part 1: $\mathbf{E}[|Q|]$ on Random Inputs

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

Lemma

$\mathbf{E}[|Q|] = 1$, *where the expectation is over random permutations* R, S, T.

Proof.

Note: $\forall i,j,k \in [n], \mathbf{P}((i,j) \in R) = \mathbf{P}((j,k) \in S) = \mathbf{P}((k,i) \in T) = 1/n$.

$$\mathbf{E}[|Q|] = \sum_{i,j,k} \mathbf{P}((i,j) \in R \wedge (j,k) \in S \wedge (k,i) \in T) \qquad \text{independence}$$

$$= \sum_{i,j,k} \mathbf{P}((i,j) \in R) \cdot \mathbf{P}((j,k) \in S) \cdot \mathbf{P}((k,i) \in T) = n^3 \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot \frac{1}{n} = 1$$

□

Proof – Part 1: $\mathbf{E}[|Q|]$ on Random Inputs

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

Lemma

$\mathbf{E}[|Q|] = 1$, *where the expectation is over random permutations* R, S, T.

Proof.

Note: $\forall i,j,k \in [n], \mathbf{P}((i,j) \in R) = \mathbf{P}((j,k) \in S) = \mathbf{P}((k,i) \in T) = 1/n$.

$$\mathbf{E}[|Q|] = \sum_{i,j,k} \mathbf{P}((i,j) \in R \wedge (j,k) \in S \wedge (k,i) \in T) \qquad \text{independence}$$

$$= \sum_{i,j,k} \mathbf{P}((i,j) \in R) \cdot \mathbf{P}((j,k) \in S) \cdot \mathbf{P}((k,i) \in T) = n^3 \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot \frac{1}{n} = 1$$

Proof – Part 1: $\mathbf{E}[|Q|]$ on Random Inputs

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

Lemma

$\mathbf{E}[|Q|] = 1$, *where the expectation is over random permutations* R, S, T.

Proof.

Note: $\forall i,j,k \in [n], \mathbf{P}((i,j) \in R) = \mathbf{P}((j,k) \in S) = \mathbf{P}((k,i) \in T) = 1/n$.

$$\mathbf{E}[|Q|] = \sum_{i,j,k} \mathbf{P}((i,j) \in R \wedge (j,k) \in S \wedge (k,i) \in T) \qquad \text{independence}$$

$$= \sum_{i,j,k} \mathbf{P}((i,j) \in R) \cdot \mathbf{P}((j,k) \in S) \cdot \mathbf{P}((k,i) \in T) = \quad n^3 \cdot \frac{1}{n} \cdot \frac{1}{n} \cdot \frac{1}{n} = 1$$

$\square$

Discussion

In expectation, there is one triangle! $\mathbf{E}[|Q|] = 1$

Will prove: if algorithm A has load $L < L_{\text{lower}}$, then $\mathbf{E}[A] = (L/L_{\text{lower}})^{3/2}$.

Proof – Part 2: What We Learn from a Message

Fix a server $u \in [p]$, and a message $\mathtt{msg}_1$ it received about R.

Definition

The set of *known tuples* is:

$$K_1(\mathtt{msg}_1) = \{(i,j) \mid \forall R(\mathtt{msg}_1(R) = \mathtt{msg}_1 \Rightarrow (i,j) \in R)\}$$

Similarly $K_2(\mathtt{msg}_2)$, $K_3(\mathtt{msg}_3)$ known tuples in S and T.

Observation

Upon receiving msg_1, msg_2, msg_3, server u can output triangle (i,j,k) iff

$$(i,j) \in K_1(msg_1) \quad \wedge \quad (j,k) \in K_2(msg_2) \quad \wedge \quad (k,i) \in K_2(msg_3)$$

Proof – Part 2: What We Learn from a Message

Fix a server $u \in [p]$, and a message $\mathtt{msg}_1$ it received about R.

Definition

The set of *known tuples* is:

$$K_1(\mathtt{msg}_1) = \{(i,j) \mid \forall R(\mathtt{msg}_1(R) = \mathtt{msg}_1 \Rightarrow (i,j) \in R)\}$$

Similarly $K_2(\mathtt{msg}_2)$, $K_3(\mathtt{msg}_3)$ known tuples in S and T.

Observation

Upon receiving msg_1, msg_2, msg_3, server u can output triangle (i,j,k) iff

$$(i,j) \in K_1(msg_1) \quad \wedge \quad (j,k) \in K_2(msg_2) \quad \wedge \quad (k,i) \in K_2(msg_3)$$

Proof – Part 2: What We Learn from a Message

Fix a server $u \in [p]$, and a message $\mathtt{msg}_1$ it received about R.

Definition

The set of *known tuples* is:

$$K_1(\mathtt{msg}_1) = \{(i,j) \mid \forall R(\mathtt{msg}_1(R) = \mathtt{msg}_1 \Rightarrow (i,j) \in R)\}$$

Similarly $K_2(\mathtt{msg}_2)$, $K_3(\mathtt{msg}_3)$ known tuples in S and T.

Observation

Upon receiving msg_1, msg_2, msg_3, server u can output triangle (i,j,k) iff

$$(i,j) \in K_1(msg_1) \quad \wedge \quad (j,k) \in K_2(msg_2) \quad \wedge \quad (k,i) \in K_2(msg_3)$$

Discussion

The useful information we learn from a message is the set of known tuples.

We show next: if $L_1 = \text{size}(\texttt{msg}_1)$ is small, then $K_1(\texttt{msg}_1)$ is small

"If you know only a few bits, then you know only a few tuples"

Later: if K_1, K_2, K_3 are small, the server knows only few triangles (AGM)

Proof – Part 3: With Few Bits You Know Only Few Tuples

Denote $H(R) = \log(n!)$ the entropy of R.

Proposition

Let $f_1 < 1$. If $msg_1(R)$ has $\leq f_1 \cdot H(R)$ bits, then $\mathbf{E}[|K_1(msg_1(R))|] \leq f_1 \cdot n$, where the expectation is over random permutations R.

Proof – Part 3: With Few Bits You Know Only Few Tuples

Denote $H(R) = \log(n!)$ the entropy of R.

Proposition

Let $f_1 < 1$. If $msg_1(R)$ has $\leq f_1 \cdot H(R)$ bits, then $\mathbf{E}[|K_1(msg_1(R))|] \leq f_1 \cdot n$, where the expectation is over random permutations R.

Proof.

Denote $k = |K_1(m_1)|$.

Proof – Part 3: With Few Bits You Know Only Few Tuples

Denote $H(R) = \log(n!)$ the entropy of R.

Proposition

Let $f_1 < 1$. If $msg_1(R)$ has $\leq f_1 \cdot H(R)$ bits, then $\mathbf{E}[|K_1(msg_1(R))|] \leq f_1 \cdot n$, where the expectation is over random permutations R.

Proof.

Denote $k = |K_1(m_1)|$.
$H(R|m_1) \leq \log[(n-k)!] \leq (1 - \frac{k}{n}) \cdot H(R)$ because $\log[(n-k)!]/(n-k) \leq \log(n!)/n$.

Proof – Part 3: With Few Bits You Know Only Few Tuples

Denote $H(R) = \log(n!)$ the entropy of R.

Proposition

Let $f_1 < 1$. If $msg_1(R)$ has $\leq f_1 \cdot H(R)$ bits, then $\mathbf{E}[|K_1(msg_1(R))|] \leq f_1 \cdot n$, where the expectation is over random permutations R.

Proof.

Denote $k = |K_1(m_1)|$.
$H(R|m_1) \leq \log[(n-k)!] \leq (1 - \frac{k}{n}) \cdot H(R)$ because $\log[(n-k)!]/(n-k) \leq \log(n!)/n$.

$H(R) = \quad H(R, \mathtt{msg}_1(R))$ $\qquad$ R determines $\mathtt{msg}_1(R)$

Proof – Part 3: With Few Bits You Know Only Few Tuples

Denote $H(R) = \log(n!)$ the entropy of R.

Proposition

Let $f_1 < 1$. If $msg_1(R)$ has $\leq f_1 \cdot H(R)$ bits, then $\mathbf{E}[|K_1(msg_1(R))|] \leq f_1 \cdot n$, where the expectation is over random permutations R.

Proof.

Denote $k = |K_1(m_1)|$.
$H(R|m_1) \leq \log[(n-k)!] \leq (1 - \frac{k}{n}) \cdot H(R)$ because $\log[(n-k)!]/(n-k) \leq \log(n!)/n$.

$$H(R) = H(R, \mathtt{msg}_1(R)) \qquad R \text{ determines } \mathtt{msg}_1(R)$$

$$= H(\mathrm{msg}_1(R)) + \textstyle\sum_{m_1} H(R|m_1) \cdot \mathbf{P}(m_1) \qquad \text{chain rule}$$

Proof – Part 3: With Few Bits You Know Only Few Tuples

Denote $H(R) = \log(n!)$ the entropy of R.

Proposition

Let $f_1 < 1$. If $msg_1(R)$ has $\leq f_1 \cdot H(R)$ bits, then $\mathbf{E}[|K_1(msg_1(R))|] \leq f_1 \cdot n$, where the expectation is over random permutations R.

Proof.

Denote $k = |K_1(m_1)|$.
$H(R|m_1) \leq \log[(n-k)!] \leq (1 - \frac{k}{n}) \cdot H(R)$ because $\log[(n-k)!]/(n-k) \leq \log(n!)/n$.

$$
\begin{aligned}
H(R) = \quad & H(R, \mathtt{msg}_1(R)) && R \text{ determines } \mathtt{msg}_1(R) \\
= \quad & H(\mathrm{msg}_1(R)) + \textstyle\sum_{m_1} H(R|m_1) \cdot \mathbf{P}(m_1) && \text{chain rule} \\
\leq \quad & f_1 \cdot H(R) + \textstyle\sum_{m_1} (1 - \frac{|K_1(m_1)|}{n}) \cdot H(R) \cdot \mathbf{P}(m_1) && \\
= \quad & f_1 \cdot H(R) + [1 - \textstyle\sum_{m_1} \frac{|K_1(m_1)| \cdot \mathbf{P}(m_1)}{n}] \cdot H(R) && = f_1 \cdot H(R) + [1 - \frac{\mathbf{E}[|K_1(m_1)|]}{n}] \cdot H(R)
\end{aligned}
$$

Proof – Part 3: With Few Bits You Know Only Few Tuples

Denote $H(R) = \log(n!)$ the entropy of R.

Proposition

Let $f_1 < 1$. If $msg_1(R)$ has $\leq f_1 \cdot H(R)$ bits, then $\mathbf{E}[|K_1(msg_1(R))|] \leq f_1 \cdot n$, where the expectation is over random permutations R.

Proof.

Denote $k = |K_1(m_1)|$.
$H(R|m_1) \leq \log[(n-k)!] \leq (1 - \frac{k}{n}) \cdot H(R)$ because $\log[(n-k)!]/(n-k) \leq \log(n!)/n$.

$$
\begin{aligned}
H(R) = \quad & H(R, \mathtt{msg}_1(R)) && R \text{ determines } \mathtt{msg}_1(R) \\
= \quad & H(\mathrm{msg}_1(R)) + \textstyle\sum_{m_1} H(R|m_1) \cdot \mathbf{P}(m_1) && \text{chain rule} \\
\leq \quad & f_1 \cdot H(R) + \textstyle\sum_{m_1} (1 - \frac{|K_1(m_1)|}{n}) \cdot H(R) \cdot \mathbf{P}(m_1) && \\
= \quad & f_1 \cdot H(R) + [1 - \textstyle\sum_{m_1} \frac{|K_1(m_1)| \cdot \mathbf{P}(m_1)}{n}] \cdot H(R) && = f_1 \cdot H(R) + [1 - \frac{\mathbf{E}[|K_1(m_1)|]}{n}] \cdot H(R)
\end{aligned}
$$

It follows: $\boxed{\mathbf{E}[|K_1(m_1)|] \leq f_1 n}$ □

Proof – Part 4: Few Known Tuples form Few Triangles

Continue to fix one server u. Its load $L = L_1 + L_2 + L_3$.

Denote: $a_{ij} = \Pr((i,j) \in K_1(\text{msg}_1(R)))$; similarly b_{jk}, c_{ki} for S, T.

Denote A_u the set of triangles returned by u:

$\mathbf{E}[|A_u|] = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} \leq \left((\sum_{ij} a_{ij}^2)(\sum_{jk} b_{jk}^2)(\sum_{ki} c_{ki}^2)\right)^{1/2}$ (Friedgut)

$a_{ij} \leq 1/n$ (why?) and $\sum_{ij} a_{ij} = \mathbf{E}[|K_1(\text{msg}_1(R))|] \leq \frac{L_1}{\log(n!)} n$ (why?)

It follows: $\sum_{ij} a_{ij}^2 \leq 1/n \sum_{ij} a_{ij} \leq \frac{L_1}{\log(n!)}$

$\mathbf{E}[|A_u] \leq \left(\frac{L_1}{\log(n!)} \cdot \frac{L_2}{\log(n!)} \cdot \frac{L_3}{\log(n!)}\right)^{1/2} \leq \left(\frac{L_1+L_2+L_3}{3\log(n!)}\right)^{3/2} = \left(\frac{L}{M}\right)^{3/2}$ triangles.

All p servers return $\mathbf{E}[|A|] \leq p\left(\frac{L}{M}\right)^{3/2} = \left(\frac{L}{\frac{M}{p^{2/3}}}\right)^{3/2} = \left(\frac{L}{L_{\text{lower}}}\right)^{3/2}$ triangles.

QED

Proof – Part 4: Few Known Tuples form Few Triangles

Continue to fix one server u. Its load $L = L_1 + L_2 + L_3$.

Denote: $a_{ij} = \Pr((i,j) \in K_1(\mathrm{msg}_1(R)))$; similarly b_{jk}, c_{ki} for S, T.

Denote A_u the set of triangles returned by u:

$\mathbf{E}[|A_u|] = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} \le \left((\sum_{ij} a_{ij}^2)(\sum_{jk} b_{jk}^2)(\sum_{ki} c_{ki}^2) \right)^{1/2}$ (Friedgut)

$a_{ij} \le 1/n$ (why?) and $\sum_{ij} a_{ij} = \mathbf{E}[|K_1(\mathrm{msg}_1(R))|] \le \frac{L_1}{\log(n!)} n$ (why?)

It follows: $\sum_{ij} a_{ij}^2 \le 1/n \sum_{ij} a_{ij} \le \frac{L_1}{\log(n!)}$

$\mathbf{E}[|A_u] \le \left(\frac{L_1}{\log(n!)} \cdot \frac{L_2}{\log(n!)} \cdot \frac{L_3}{\log(n!)} \right)^{1/2} \le \left(\frac{L_1+L_2+L_3}{3\log(n!)} \right)^{3/2} = \left(\frac{L}{M} \right)^{3/2}$ triangles.

All p servers return $\mathbf{E}[|A|] \le p \left(\frac{L}{M} \right)^{3/2} = \left(\frac{L}{\frac{M}{p^{2/3}}} \right)^{3/2} = \left(\frac{L}{L_{\text{lower}}} \right)^{3/2}$ triangles.

QED

Proof – Part 4: Few Known Tuples form Few Triangles

Continue to fix one server u. Its load $L = L_1 + L_2 + L_3$.

Denote: $a_{ij} = \Pr((i,j) \in K_1(\text{msg}_1(R)))$; similarly b_{jk}, c_{ki} for S, T.

Denote A_u the set of triangles returned by u:

$$\mathbf{E}[|A_u|] = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} \leq \left((\sum_{ij} a_{ij}^2)(\sum_{jk} b_{jk}^2)(\sum_{ki} c_{ki}^2) \right)^{1/2} \qquad \text{(Friedgut)}$$

$a_{ij} \leq 1/n$ (why?) and $\sum_{ij} a_{ij} = \mathbf{E}[|K_1(\text{msg}_1(R))|] \leq \frac{L_1}{\log(n!)} n$ (why?)

It follows: $\sum_{ij} a_{ij}^2 \leq 1/n \sum_{ij} a_{ij} \leq \frac{L_1}{\log(n!)}$

$$\mathbf{E}[|A_u|] \leq \left(\frac{L_1}{\log(n!)} \cdot \frac{L_2}{\log(n!)} \cdot \frac{L_3}{\log(n!)} \right)^{1/2} \leq \left(\frac{L_1 + L_2 + L_3}{3 \log(n!)} \right)^{3/2} = \left(\frac{L}{M} \right)^{3/2} \text{ triangles.}$$

All p servers return $\mathbf{E}[|A|] \leq p \left(\frac{L}{M} \right)^{3/2} = \left(\frac{L}{\frac{M}{p^{2/3}}} \right)^{3/2} = \left(\frac{L}{L_{\text{lower}}} \right)^{3/2}$ triangles.

QED

Proof – Part 4: Few Known Tuples form Few Triangles

Continue to fix one server u. Its load $L = L_1 + L_2 + L_3$.

Denote: $a_{ij} = \Pr((i,j) \in K_1(\text{msg}_1(R)))$; similarly b_{jk}, c_{ki} for S, T.

Denote A_u the set of triangles returned by u:

$$\mathbf{E}[|A_u|] = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} \leq \left((\sum_{ij} a_{ij}^2)(\sum_{jk} b_{jk}^2)(\sum_{ki} c_{ki}^2)\right)^{1/2} \quad \text{(Friedgut)}$$

$a_{ij} \leq 1/n$ (why?) and $\sum_{ij} a_{ij} = \mathbf{E}[|K_1(\text{msg}_1(R))|] \leq \frac{L_1}{\log(n!)} n$ (why?)

It follows: $\sum_{ij} a_{ij}^2 \leq 1/n \sum_{ij} a_{ij} \leq \frac{L_1}{\log(n!)}$

$$\mathbf{E}[|A_u] \leq \left(\frac{L_1}{\log(n!)} \cdot \frac{L_2}{\log(n!)} \cdot \frac{L_3}{\log(n!)}\right)^{1/2} \leq \left(\frac{L_1+L_2+L_3}{3\log(n!)}\right)^{3/2} = \left(\frac{L}{M}\right)^{3/2} \text{ triangles.}$$

All p servers return $\mathbf{E}[|A|] \leq p\left(\frac{L}{M}\right)^{3/2} = \left(\frac{L}{\frac{M}{p^{2/3}}}\right)^{3/2} = \left(\frac{L}{L_{\text{lower}}}\right)^{3/2}$ triangles.

QED

Proof – Part 4: Few Known Tuples form Few Triangles

Continue to fix one server u. Its load $L = L_1 + L_2 + L_3$.

Denote: $a_{ij} = \Pr((i,j) \in K_1(\text{msg}_1(R)))$; similarly b_{jk}, c_{ki} for S, T.

Denote A_u the set of triangles returned by u:

$$\mathbf{E}[|A_u|] = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} \leq \left((\sum_{ij} a_{ij}^2)(\sum_{jk} b_{jk}^2)(\sum_{ki} c_{ki}^2)\right)^{1/2} \qquad \text{(Friedgut)}$$

$a_{ij} \leq 1/n$ (why?) and $\sum_{ij} a_{ij} = \mathbf{E}[|K_1(\text{msg}_1(R))|] \leq \frac{L_1}{\log(n!)} n$ (why?)

It follows: $\sum_{ij} a_{ij}^2 \leq 1/n \sum_{ij} a_{ij} \leq \frac{L_1}{\log(n!)}$

$$\mathbf{E}[|A_u|] \leq \left(\frac{L_1}{\log(n!)} \cdot \frac{L_2}{\log(n!)} \cdot \frac{L_3}{\log(n!)}\right)^{1/2} \leq \left(\frac{L_1+L_2+L_3}{3\log(n!)}\right)^{3/2} = \left(\frac{L}{M}\right)^{3/2} \text{ triangles.}$$

All p servers return $\mathbf{E}[|A|] \leq p\left(\frac{L}{M}\right)^{3/2} = \left(\frac{L}{\frac{M}{p^{2/3}}}\right)^{3/2} = \left(\frac{L}{L_{\text{lower}}}\right)^{3/2}$ triangles.

QED

Proof – Part 4: Few Known Tuples form Few Triangles

Continue to fix one server u. Its load $L = L_1 + L_2 + L_3$.

Denote: $a_{ij} = \Pr((i,j) \in K_1(\text{msg}_1(R)))$; similarly b_{jk}, c_{ki} for S, T.

Denote A_u the set of triangles returned by u:

$$\mathbf{E}[|A_u|] = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} \le \left((\sum_{ij} a_{ij}^2)(\sum_{jk} b_{jk}^2)(\sum_{ki} c_{ki}^2) \right)^{1/2} \quad \text{(Friedgut)}$$

$a_{ij} \le 1/n$ (why?) and $\sum_{ij} a_{ij} = \mathbf{E}[|K_1(\text{msg}_1(R))|] \le \frac{L_1}{\log(n!)} n$ (why?)

It follows: $\sum_{ij} a_{ij}^2 \le 1/n \sum_{ij} a_{ij} \le \frac{L_1}{\log(n!)}$

$$\mathbf{E}[|A_u] \le \left(\frac{L_1}{\log(n!)} \cdot \frac{L_2}{\log(n!)} \cdot \frac{L_3}{\log(n!)} \right)^{1/2} \le \left(\frac{L_1+L_2+L_3}{3\log(n!)} \right)^{3/2} = \left(\frac{L}{M} \right)^{3/2} \text{ triangles.}$$

All p servers return $\mathbf{E}[|A|] \le p \left(\frac{L}{M} \right)^{3/2} = \left(\frac{L}{\frac{M}{p^{2/3}}} \right)^{3/2} = \left(\frac{L}{L_{\text{lower}}} \right)^{3/2}$ triangles.

QED

Proof – Part 4: Few Known Tuples form Few Triangles

Continue to fix one server u. Its load $L = L_1 + L_2 + L_3$.

Denote: $a_{ij} = \Pr((i,j) \in K_1(\text{msg}_1(R)))$; similarly b_{jk}, c_{ki} for S, T.

Denote A_u the set of triangles returned by u:

$\mathbf{E}[|A_u|] = \sum_{i,j,k} a_{ij} b_{jk} c_{ki} \leq \left((\sum_{ij} a_{ij}^2)(\sum_{jk} b_{jk}^2)(\sum_{ki} c_{ki}^2)\right)^{1/2}$ (Friedgut)

$a_{ij} \leq 1/n$ (why?) and $\sum_{ij} a_{ij} = \mathbf{E}[|K_1(\text{msg}_1(R))|] \leq \frac{L_1}{\log(n!)} n$ (why?)

It follows: $\sum_{ij} a_{ij}^2 \leq 1/n \sum_{ij} a_{ij} \leq \frac{L_1}{\log(n!)}$

$\mathbf{E}[|A_u] \leq \left(\frac{L_1}{\log(n!)} \cdot \frac{L_2}{\log(n!)} \cdot \frac{L_3}{\log(n!)}\right)^{1/2} \leq \left(\frac{L_1+L_2+L_3}{3\log(n!)}\right)^{3/2} = \left(\frac{L}{M}\right)^{3/2}$ triangles.

All p servers return $\mathbf{E}[|A|] \leq p\left(\frac{L}{M}\right)^{3/2} = \left(\frac{L}{\frac{M}{p^{2/3}}}\right)^{3/2} = \left(\frac{L}{L_{\text{lower}}}\right)^{3/2}$ triangles.

QED

Discussion of the Lower Bound for the Triangle Query

Algorithm A with load $L < L_{\text{lower}} = m/p^{2/3}$, $\mathbf{E}[A] \leq (L/L_{\text{lower}})^{3/2}\mathbf{E}[|Q|]$

- The result is for Skew-free Data
- There exists at least one instance on which A fails (trivial).
- Even if A is randomized, there exists at least one instance where A fails with high probability (Yao's lemma).
- Parallelism gets harder as p increases. An algorithm with load $L = O(\frac{m}{p^{1-\varepsilon}})$, with $\varepsilon < 1/3$ reports only $O(\frac{1}{p^{1/3-\varepsilon}})$ triangles. Fewer, as p increases!

Discussion of the Lower Bound for the Triangle Query

Algorithm A with load $L < L_{\text{lower}} = m/p^{2/3}$, $\mathbf{E}[A] \le (L/L_{\text{lower}})^{3/2}\mathbf{E}[|Q|]$

- The result is for Skew-free Data
- There exists at least one instance on which A fails (trivial).
- Even if A is randomized, there exists at least one instance where A fails with high probability (Yao's lemma).
- Parallelism gets harder as p increases. An algorithm with load $L = O(\frac{m}{p^{1-\varepsilon}})$, with $\varepsilon < 1/3$ reports only $O(\frac{1}{p^{1/3-\varepsilon}})$ triangles. Fewer, as p increases!

Generalization to Full Conjunctive Queries

- Will discuss the equal-cardinality case today.
- Will discuss the general case in Lecture 3.

The Fractional Vertex Cover / Edge Packing

Hypergraph: $Q = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$ Nodes: $x_1, \ldots, x_k$, edges: $R_1, \ldots, R_\ell$.

Definition

A *fractional vertex cover* of Q is a sequence $v_1 \geq 0, \ldots, v_k \geq 0$ such that:

$$\forall j : \sum_{i : x_i \in R_j} v_i \geq 1$$

A *fractional edge packing* of Q is a sequence $u_1 \geq 0, \ldots, u_\ell \geq 0$ such that:

$$\forall i : \sum_{j : x_i \in R_j} u_j \leq 1$$

By duality: $\min_{\mathbf{v}} \sum_i v_i = \max_{\mathbf{u}} \sum_j u_j = \tau^*$

The Fractional Vertex Cover / Edge Packing

Hypergraph: $Q = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell)$ Nodes: $x_1, \ldots, x_k$, edges: $R_1, \ldots, R_\ell$.

Definition

A *fractional vertex cover* of Q is a sequence $v_1 \geq 0, \ldots, v_k \geq 0$ such that:

$$\forall j : \sum_{i : x_i \in R_j} v_i \geq 1$$

A *fractional edge packing* of Q is a sequence $u_1 \geq 0, \ldots, u_\ell \geq 0$ such that:

$$\forall i : \sum_{j : x_i \in R_j} u_j \leq 1$$

By duality: $\min_{\mathbf{v}} \sum_i v_i = \max_{\mathbf{u}} \sum_j u_j = \tau^*$

The HyperCube Algorithm for General Queries

$Q(\mathbf{x}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell), \quad |R_1| = \ldots = |R_\ell| = m$

p servers.

The HyperCube Algorithm for General Queries

$Q(\mathbf{x}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell), \quad |R_1| = \ldots = |R_\ell| = m$

p servers.

$\mathbf{v} = (v_1, \ldots, v_k)$ any fractional vertex cover; $v_0 \overset{\text{def}}{=} \sum_i v_i$.

Organize the p servers in a hypercube: $[p] \equiv [p^{\frac{v_1}{v_0}}] \times \cdots \times [p^{\frac{v_k}{v_0}}]$.

Choose k independent hash functions $h_1, \ldots, h_k$

The HyperCube Algorithm for General Queries

$Q(\mathbf{x}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell), \quad |R_1| = \ldots = |R_\ell| = m$

p servers.

$\mathbf{v} = (v_1, \ldots, v_k)$ any fractional vertex cover; $v_0 \stackrel{\text{def}}{=} \sum_i v_i$.

Organize the p servers in a hypercube: $[p] \equiv [p^{\frac{v_1}{v_0}}] \times \cdots \times [p^{\frac{v_k}{v_0}}]$.

Choose k independent hash functions $h_1, \ldots, h_k$

Round 1: Each server sends each tuple $R_j(x_{j_1}, x_{j_2}, \ldots)$ to all servers whose coordinates $j_1, j_2, \ldots$ are $h_{j_1}(x_{j_1}), h_{j_2}(x_{j_2}), \ldots$ and broadcasts along the missing dimensions. Then, each server computes Q on its local data.

The HyperCube Algorithm for General Queries

$Q(\mathbf{x}) = R_1(\mathbf{x}_1), \ldots, R_\ell(\mathbf{x}_\ell), \quad |R_1| = \ldots = |R_\ell| = m$

p servers.

$\mathbf{v} = (v_1, \ldots, v_k)$ any fractional vertex cover; $v_0 \stackrel{\text{def}}{=} \sum_i v_i$.

Organize the p servers in a hypercube: $[p] \equiv [p^{\frac{v_1}{v_0}}] \times \cdots \times [p^{\frac{v_k}{v_0}}]$.

Choose k independent hash functions $h_1, \ldots, h_k$

Round 1 Each server sends each tuple $R_j(x_{j_1}, x_{j_2}, \ldots)$ to all servers whose coordinates $j_1, j_2, \ldots$ are $h_{j_1}(x_{j_1}), h_{j_2}(x_{j_2}), \ldots$ and broadcasts along the missing dimensions. Then, each server computes Q on its local data.

Theorem

The load of the HyperCube algorithm is $L = \ell \frac{m}{p^{1/v_0}} = O(\frac{m}{p^{1/v_0}})$.

Proof.

Fix a server. $\mathbf{E}[\#\text{ tuples in } R_j] = \frac{m}{p^{\sum_{i \in R_j} v_i / v_0}} \le \frac{m}{p^{1/v_0}}$ since $\sum_{i \in R_j} v_i \ge 1$. □

Lower Bound for General Queries

Definition

$R \subseteq [n]^r$ is called a *matching* of arity r if $|R| = n$ and every column is a key.

Example: matching of arity 3, $n = 4$

R

x	y	z
1	3	2
2	1	4
3	4	3
4	2	1

Theorem

Suppose all arities are ≥ 2. For any packing $\mathbf{u}$, denote $L_{lower} = \frac{m}{p^{1/\sum_j u_j}}$. Then, for any algorithm A with load $L < L_{lower}$, it returns $\mathbf{E}[|A|] \leq (L/L_{lower})^u \mathbf{E}[|Q|]$ answers, where the expectation is over random matchings

Proof in the section today. (Q: what about arities 1?)

Lower Bound for General Queries

Corollary

HyperCube algorithm is optimal. (Because $\min_{\mathbf{v}} \sum_i v_i = \max_{\mathbf{u}} \sum_j u_j = \tau^*$*.)*

Summary of Lecture 2

- Parallel query engines today compute one join at a time.
  Issues: communication rounds, intermediate results, skew.
- The HyperCube algorithm: one round. Restrictions:
  Equal-cardinalities (will generalize in Lecture 3)
  Skew-free databases (skew is open, but will discuss in Lecture 4).
- HyperCube is more resilient to skew than a join.
- A surprising fact:
  Parallel algorithms: lower bound given by *fractional edge packing*.
  Sequential algorithms: lower given by *fractional edge cover*.
- Multiple rounds: all lower bounds [Beame'13] use weaker model.
  Open problem: an algorithm with load $O(n/p)$ cannot compute
  $$Q = R_1(x_0, x_1), R_2(x_1, x_2), R_3(x_2, x_3), R_4(x_3, x_4), R_5(x_4, x_5)$$
  in two rounds, over random permutations.