DATA516/CSED516
Scalable Data Systems and Algorithms
Lecture 6
Parallel Execution Wrapup
Graph Processing
Announcements

• Feedback for project proposals

• Redshift + Snowflake review was due

• AWS Academy class

• HW3 due on Monday
Today

- Skew
- Parallel Query Processing Wrap-up
- Graphs, datalog
Skew
Skew

- Skew means that one server runs much longer than the other servers

- Reasons:
  - Computation skew
  - Data skew
Data Skew

Assume we hash-partition data items

• All records with same partition key are sent to the same server

• Heavy hitter
Analyzing Heavy Hitters

• How many times can an item occur before it is called a “heavy hitter”?

• The answer requires a deep analysis of what a good hash function can do.
Problem Statement

Given: \( N \) distinct data items \( v_1, \ldots, v_N \)

- We hash-partition them to \( P \) nodes
- How uniform is the partition?
Discussion

There is no deterministic “good” hash function:

- For any hash function $h$, there exists distinct data items $v_1, ..., v_N$ s.t. all are mapped to the same value:
  $h(v_1)=h(v_2)= ... h(v_N)$
Discussion

There is no deterministic “good” hash function:

• For any hash function \( h \), there exists distinct data items \( v_1, \ldots, v_N \) s.t. all are mapped to the same value:
  \[
  h(v_1)=h(v_2)= \ldots h(v_N)
  \]

A “good” hash function is random function:

• Intuition: every day we choose another \( h \), we want the partition to be uniform in expectation
Balls into Bins

Probability of a ball getting into bin #5:

We throw $N$ balls randomly into $P$ bins:

- When $N \gg P$ then
- When $N = P$ then $N \log N$
Balls into Bins

We throw $N$ balls randomly into $P$ bins:

Probability of a ball getting into bin #5: \( \frac{1}{P} \)

Expected size of bin #5:

Expected size of the max bin:

• When $N \gg P$ then \( \Omega \)

• When $N = P$ then \( N \log N \)
Balls into Bins

Probability of a ball getting into bin #5: \( \frac{1}{P} \)

Expected size of bin #5:

We throw N balls randomly into P bins:
Balls into Bins

We throw $N$ balls randomly into $P$ bins:

- Probability of a ball getting into bin #5:
  - $\frac{1}{P}$

- Expected size of bin #5:
  - $\frac{N}{P}$

- Expected size of the max bin:
  - $\mathcal{O}(\log N)$

When $N \gg P$ then

When $N = P$ then

$\frac{N}{P}$
Balls into Bins

We throw N balls randomly into P bins:

Probability of a ball getting into bin #5:

Expected size of bin #5:

Expected size of the max bin:

1. When $N \gg P$ then $O(1)$
2. When $N = P$ then $\frac{N}{\log N}$
Balls into Bins

We throw $N$ balls randomly into $P$ bins:

- Probability of a ball getting into bin #5: $\frac{1}{P}$
- Expected size of bin #5:
- Expected size of the max bin:
  - When $N \gg P$ then $\mathcal{O}\left(\frac{N}{P}\right)$
Balls into Bins

We throw $N$ balls randomly into $P$ bins:

- Probability of a ball getting into bin #5:
- Expected size of bin #5:
- Expected size of the max bin:
  - When $N \gg P$ then $0 \left( \frac{N}{P} \right)$
  - When $N=P$ then $\log N$
Discussion

• Hash-partition is like throwing $N$ balls into $P$ bins

• Partition is uniform when max load is approx $N/P$

• To analyze the max load, let's analyze the load of one fixed bin
One Bin

$N$ data items $v_1, \ldots, v_N$

One fixed bin
One Bin

$N$ data items $v_1, \ldots, v_N$

For each data item $v_i$, let $X_i$ be a r.v. s.t.:

$X_i = 1$ if item $v_i$ is sent to our bin
$X_i = 0$ if item $v_i$ is sent to a different bin
One Bin

N data items $v_1, \ldots, v_N$

For each data item $v_i$, let $X_i$ be a r.v. s.t.:

- $X_i = 1$ if item $v_i$ is sent to our bin
- $X_i = 0$ if item $v_i$ is sent to a different bin

$X_i \in \{0, 1\}$ is a Bernoulli random variable

$\Pr(X_i = 1) = 1/P$
One Bin

$N$ data items $v_1, \ldots, v_N$

For each data item $v_i$, let $X_i$ be a r.v. s.t.:

- $X_i = 1$ if item $v_i$ is sent to our bin
- $X_i = 0$ if item $v_i$ is sent to a different bin

$X_i \in \{0,1\}$ is a Bernoulli random variable

$\Pr(X_i = 1) = 1/P$

Load of the bin is: $Y = X_1 + X_2 + \cdots + X_N$
The Cernoff Bound

Bernoulli r.v.: \( X_1, \ldots, X_N \in \{0,1\} \)

For all \( i \), \( \Pr(X_i = 1) = \mu \in (0,1) \)

We are interested in \( Y = X_1 + X_2 + \cdots + X_N \)

Note: very many variants
The Cernoff Bound

Bernoulli r.v.: $X_1, \ldots, X_N \in \{0,1\}$

For all $i$, $\Pr(X_i = 1) = \mu \in (0,1)$

We are interested in $Y = X_1 + X_2 + \cdots + X_N$

**Fact:** $E[Y] = N\mu$

Note: very many variants
The Cernoff Bound

Bernoulli r.v.: \( X_1, \ldots, X_N \in \{0,1\} \)

For all \( i \), \( \Pr(X_i = 1) = \mu \in (0,1) \)

We are interested in \( Y = X_1 + X_2 + \cdots + X_N \)

**Fact:** \( E[Y] = N\mu \)

**Theorem** (Cernoff bound). If they are iid then:

\[
\Pr(Y > (1 + \delta)E[Y]) \leq \exp\left(-\frac{\delta^2}{3}E[Y]\right)
\]
Application to Hash Partition

N data items $v_1, \ldots, v_N$
Distribute on $P$ servers

Fix one server $j$
Indicator variables:

$$X_i = [h(v_i) = j]$$
$$\Pr(X_i = 1) = 1/P$$
Application to Hash Partition

Load of server $j$: $\text{Load}(j) = X_1 + X_2 + \cdots + X_N$

Expected load: $E[\text{Load}(j)] = \frac{N}{P}$
Application to Hash Partition

Load of server $j$: $\text{Load}(j) = X_1 + X_2 + \cdots + X_N$

Expected load: $\mathbb{E}[\text{Load}(j)] = \frac{N}{P}$

Cernoff: $\Pr \left( \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq \exp \left( -\frac{\delta^2 N}{3P} \right)$
Application to Hash Partition

**Load of server** \( j \): \( 
\text{Load}(j) = X_1 + X_2 + \cdots + X_N
\)

**Expected load:** \( 
\mathbb{E}[\text{Load}(j)] = \frac{N}{P}
\)

**Cernoff:** \( 
\Pr\left( \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq \exp\left( -\frac{\delta^2 N}{3 P} \right)
\)

**Union bound:** \( 
\Pr(\text{Skew}) \leq P \cdot \exp\left( -\frac{\delta^2 N}{3 P} \right)
\)
Discussion

• We have not computed the expected value of the maximum load
  – I don’t even know how to do that

• Instead, we have computed the probability that we exceed the expected load by more than delta
Example 1

\[ \Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( - \frac{\delta^2 N}{3P} \right) \]

N=2,000,000 items, P=100 servers:
We expect: L = 20,000 items/server
Example 1

\[
\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( - \frac{\delta^2 N}{3 P} \right)
\]

N=2,000,000 items, P=100 servers:
We expect: L = 20,000 items/server
What is the prob that some server exceeds L by > 10%?
Example 1

\[
\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( - \frac{\delta^2 N}{3P} \right)
\]

N=2,000,000 items, P=100 servers:
We expect: L = 20,000 items/server
What is the prob that some server exceeds L by > 10%?

\[
\Pr(\text{bad}) \leq
\]
Example 1

\[ \Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3P} \right) \]

N=2,000,000 items, P=100 servers:
We expect: L = 20,000 items/server
What is the prob that some server exceeds L by > 10%?

\[ Pr(\text{bad} ) \leq 100 \times \exp \left( -\frac{0.1^2}{3} \times 20000 \right) \]
Example 1

\[
\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( - \frac{\delta^2 N}{3P} \right)
\]

N=2,000,000 items, P=100 servers:
We expect: L = 20,000 items/server
What is the prob that some server exceeds L by > 10%?

\[
Pr(\text{bad}) \leq 100 \times \exp \left( - \frac{0.1^2}{3} \times 20000 \right) = 100 \times \exp(-66)
\]
Example 1

Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( - \frac{\delta^2 N}{3P} \right)

N=2,000,000 \text{ items, } P=100 \text{ servers:}
We expect: \( L = 20,000 \text{ items/server} \)
What is the prob that some server exceeds \( L \) by > 10%?

\[
 Pr(\text{bad}) \leq 100 \times \exp \left( - \frac{0.1^2}{3} \times 20000 \right) = 100 \times \exp(-66)
\]

Virtually zero!
Example 2

\[ \Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3P} \right) \]

N=2,000,000 items, P=10,000 servers:
Example 2

\[ \Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3P} \right) \]

N=2,000,000 items, P=10,000 servers:
We expect: L = 200 items/server
What is the prob that some server exceeds L by > 10%?
Example 2

Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot exp \left( - \frac{\delta^2 N}{3P} \right)

N=2,000,000 items, P=10,000 servers:
We expect: L = 200 items/server
What is the prob that some server exceeds L by > 10%?

Pr(bad ) \leq
Example 2

Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)

N=2,000,000 \text{ items, } P=10,000 \text{ servers: }
We expect: \text{L} = 200 \text{ items/server}
What is the prob that some server exceeds \text{L} by > 10%?

Pr(bad) \leq 10000 \times \exp \left( -\frac{0.1^2}{3} \times 200 \right)
Example 2

Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3P} \right)

N=2,000,000 \text{ items}, P=10,000 \text{ servers}:

We expect: \( L = 200 \) items/server

What is the prob that some server exceeds \( L \) by > 10%?

\[
Pr(\text{bad}) \leq 10000 \times \exp \left( -\frac{0.1^2}{3} \times 200 \right) = 10000 \times \exp \left( -\frac{2}{3} \right) = 5134
\]
Example 2

\[
\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 \frac{P}{P}} \right)
\]

N=2,000,000 items, P=10,000 servers:
We expect: L = 200 items/server
What is the prob that some server exceeds L by > 10%?

\[
Pr(\text{bad}) \leq 10000 \times \exp \left( -\frac{0.1^2}{3} \times 200 \right) = 10000 \times \exp \left( -\frac{2}{3} \right) = 5134
\]

>>1
Skew almost certain
Main Take-away

\[ \text{Pr}(\text{Skew}) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3P} \right) \]

To avoid skew, we need \( N \gg P \)
Application to Heavy Hitters

N data items $v_1, \ldots, v_N$ some are repeated

• We hash-partition them to $P$ nodes

• How uniform is the partition?
Application to Heavy Hitters

• Assume worst case: each item t times
• \( \frac{N}{t} \) distinct items

\[ v_1, v_1, ..., v_1, v_2, v_2, ..., v_2, ..., ..., v_{\frac{N}{t}}, v_{\frac{N}{t}}, ..., v_{\frac{N}{t}} \]

\[ t \quad t \quad t \]
Application to Heavy Hitters

• Assume worst case: each item $t$ times

• $N/t$ distinct items

\[ v_1, v_1, ..., v_1, v_2, v_2, ..., v_2, ..., ..., v_{N/t}, v_{N/t}, ..., v_{N/t} \]

\[ t \quad t \quad t \]

• $N/t$ iid Bernoulli variables:

\[ X_1, X_2, ..., X_{N/t}, \quad \Pr(X_i = 1) = \frac{1}{P} \]
Application to Heavy Hitters

• Assume worst case: each item $t$ times
• $N/t$ distinct items

\[
\underbrace{\nu_1, \nu_1, \ldots, \nu_1, \nu_2, \nu_2, \ldots, \nu_2, \ldots, \nu_{N/t}, \nu_{N/t}, \ldots, \nu_{N/t}}_{t \quad t \quad t}
\]

• $N/t$ iid Bernoulli variables:

\[
X_1, X_2, \ldots, X_{N/t}, \quad \Pr(X_i = 1) = \frac{1}{P}
\]

• Load at a fixed server is $t \times Y$ where:

\[
Y = (X_1 + X_2 + \cdots + X_{N/t}), \quad E[Y] = \frac{N}{tP} = \frac{N}{tP}
\]
Application to Heavy Hitters

\[ Y = (X_1 + X_2 + \cdots + X_{N/t}), \quad E[Y] = \frac{N}{tP} = \frac{N}{tP} \]
Application to Heavy Hitters

\[ Y = \left( X_1 + X_2 + \cdots + X_{N/t} \right), \quad E[Y] = \frac{N}{tP} = \frac{N}{tP} \]

We apply Cernoff:

- Skew at one, fixed server:

\[
\Pr(tY > (1 + \delta) tE[Y]) \leq \exp \left( -\frac{\delta^2}{3} E[Y] \right) = \exp \left( -\frac{\delta^2}{3} \frac{N}{tP} \right)
\]
Application to Heavy Hitters

\[ Y = \left( X_1 + X_2 + \cdots + X_{N/t} \right), \quad E[Y] = \frac{N}{tP} = \frac{N}{tP} \]

We apply Cernoff:

- Skew at one, fixed server:

\[
\Pr(tY > (1 + \delta) tE[Y]) \leq \exp \left( -\frac{\delta^2}{3} E[Y] \right) = \exp \left( -\frac{\delta^2}{3} \frac{N}{tP} \right)
\]

- Skew at any server

\[
\Pr(\text{skew}) \leq P \exp \left( -\frac{\delta^2}{3} \frac{N}{tP} \right)
\]
Main Take-away

\[ \Pr(\text{skew}) \leq P \exp \left( -\frac{\delta^2 N}{3 tP} \right) \]

To avoid skew, we need \( t << \frac{N}{P} \)
Discussion

• Many distributed query processors do not handle data skew well

• (Project idea: how does your favorite engine handle skewed data?)

• In practice, you may need to partition skewed data manually
Today

- Skew
- Parallel Query Processing Wrap-up
- Graphs, datalog
Parallel Query Processing
Wrap-up
Recap: Parallel Architectures:

1.

2.

3.
Recap: Parallel Architectures:

1. Shared Memory

2. Shared Disk

3. Shared Nothing – aka distributed
Recap: Motivation

• Discuss when to use distributed data processing v.s. single server

• [in class]
Recap: Explain these terms

- Speedup v.s. Scaleup
- Scaleup v.s. Scaleout
Recap:
Horizontal Data Partitioning

Describe three strategies:

1.

2.

3.
Recap: Horizontal Data Partitioning

Describe three strategies:

1. Block partition
2. Hash partition
3. Range partition
Recap: Distributed Join

Describe/discuss these algorithms:

1. Parallel Hash Join

2. Broadcast join, a.k.a. small join
Case study: Snowflake
Snowflake

• It is an SaaS – what is this?  Give other examples of types of cloud services…
Snowflake

• It is an SaaS – what is this? Give other examples of types of cloud services…
• SaaS = software as a service
• Other examples:
  – Platform as a service (PaaS): e.g. Amazon’s EC
  – Infrastructure as a service (virtual machines)
  – Function as a Service: Amazon’s Lambda
Snowflake

- Describe Snowflake’s Data Storage
Snowflake

- Describe Snowflake’s Data Storage

In class:
- S3: PUT/GET/DELETE
- Table → horizontal partition in files
- Blobs+PAX
- Temp storage → S3

Figure 1: Multi-Cluster, Shared Data Architecture
Snowflake

• Describe Elasticity in Snowflake

• Describe failure handling in Snowflake
Snowflake

• Describe Elasticity in Snowflake
  – Virtual Warehouse (VW) serves one user
  – T-Shirt sizes: X-Small … XX-Large
  – Small query may run on subset of VW

• Describe failure handling in Snowflake
Snowflake

• Describe Elasticity in Snowflake
  – Virtual Warehouse (VW) serves one user
  – T-Shirt sizes: X-Small … XX-Large
  – Small query may run on subset of VW

• Describe failure handling in Snowflake
  – Restart the query
  – No partial retries (like MapReduce or Spark)
Snowflake

• Describe its execution engine
Snowflake

• Describe its execution engine

• Column-oriented (in class)

• Vectorized (“tuple batches” – in class)

• Push-based (in class)
Snowflake

• What does Snowflake use instead of indexes?
Snowflake

• What does Snowflake use instead of indexes?

• “Pruning”: for each file (recall: this is a horizontal partition of a table) and each attribute, it stores the min/max values in that column in that file; may skip files when not needed.
Conclusion

• Distributed data processing:
  – Spread the data to fit in main memory
  – Take advantage of parallelism

• “SQL is embarrassingly parallel”
  – Relational algebra: easy to parallelize
  – Hash-based algorithm suffer from skew
Today

• Skew

• Parallel Query Processing Wrap-up

  • Graphs, datalog
Graphs
Graph Processing Motivation

- Many apps need to do analytics on graphs
  - Web graph
  - Social networks
  - Transportation routes
  - Citation graphs
  - Disease propagation graphs
  - ...

- A graph: G(V,E)
  - V: Vertices in the graph
  - E: Edges between the vertices
  - Large graph means many edges, not many gigabytes
Graph Analysis

- Graph analytics has several unique properties
  - One large object: the graph
  - Difficult to partition and process in parallel
  - Iterative processing
    - Little work per vertex at each iteration
    - Many iterations & significant amount of communication
- Example applications
  - Shortest path
  - Clustering
  - Page rank and variants
  - Triangles and other structure
  - ...
Example 1: Pattern Matching

Pattern

Graph
Example 1: Pattern Matching

Pattern

Graph
Example 1: Pattern Matching

Pattern

Graph
Example 1: Pattern Matching

Pattern

Graph
Example 2: Descendants

ParentOf(1,2)

Find all descendants of the red node

Recursively follow the ParentOf links until no new descendants are found
Example 3: Connected Components
Example 3: Connected Components

DATA516/CSED516 - Fall 2021
Example 3: Connected Components

1

1

1

5

1

5

1

5

DATA516/CSED516 - Fall 2021
Example 4: PageRank

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page.

Iterate until convergence

$$PR(p_i,t+1) = \frac{(1-d)}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j,t)}{L(p_j)}$$

Where

- $PR(x)$: Page rank of $x$
- $L(x)$: # Outgoing links from $x$

How to Model Graph Analytics

• **Option 1: Relational Model**
  – Relation Edges(v1,v2)
  – Optionally can also have a relation Vertices(v)
  – Relational queries

• **Option 2: Graph Model**
  – The graph is a first-class citizen
  – Vertex-based API
  – Pattern-based and/or traversal-based queries

• **Option 3: Linear Algebra**
  – Graph as a matrix
Processing Graphs in SQL

Graph

R=

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Processing Graphs in SQL

Graph

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Processing Graphs in SQL

Graph

Pattern Matching

\[ R = \]

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

SELECT ...
FROM ...
WHERE ...
Processing Graphs in SQL

R =

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

SELECT ... FROM ... WHERE ...
Processing Graphs in SQL

Graph

```
 R= 

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
```

Pattern Matching

```
 SELECT x.src, y.dst, z.dst
 FROM R x, R y, R z
 WHERE x.src = y.src AND x.dst = z.dst AND y.dst = z.src
```
Processing Graphs in SQL

Graph

R=

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE
Processing Graphs in SQL

Graph

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Pattern Matching

```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE x.src = y.src
  and x.dst = z.dst
  and y.dst = z.src
```
Processing Graphs in SQL

Graph

Pattern Matching

R=

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x.src</th>
<th>y.dst</th>
<th>z.dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE x.src = y.src
    and x.dst = z.dst
    and y.dst = z.src
```
Processing Graphs in SQL

A pattern with n edges becomes an n-way self-join

**Graph**

```
<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
```

**Pattern Matching**

```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE x.src = y.src
    and x.dst = z.dst
    and y.dst = z.src
```
Processing Graphs in SQL

Graph

Find Descendants of node 2

R=

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Processing Graphs in SQL

Graph

Find Descendants of node 2
Find children:

R=

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Processing Graphs in SQL

Graph

Find Descendants of node 2

Find children:

```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```
Processing Graphs in SQL

Graph

Find Descendants of node 2

Find children:

SELECT x.dst as d
FROM R x
WHERE x.src = 2

R=

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Processing Graphs in SQL

Graph

Find Descendants of node 2

```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

Find children:

```
UNION
SELECT DISTINCT y.dst as d
FROM R x, R y
WHERE x.src = 2 and x.dst = y.src
```
Processing Graphs in SQL

Graph

R=

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Find Descendants of node 2

Find children:

```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

...and their children

```
UNION
SELECT DISTINCT y.dst as d
FROM R x, R y
WHERE x.src = 2 and x.dst = y.src
```

...and their children...
Processing Graphs in SQL

Find Descendants of node 2

Find children:

```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

...and their children...

```
UNION
SELECT DISTINCT y.dst as d
FROM R x, R y
WHERE x.src = 2 and x.dst = y.src
```

...and their children...

```
...and their children...
```
Discussion

• Graph processing often requires recursion:
  – Descendants, connected components, etc

• SQL *does* support recursion using WITH and CTE (Common Table Expression)
  – Lots of restrictions

• Origin of recursion in SQL: datalog
Datalog

- Designed in the 80’s: simple, concise, elegant, very popular in research

- All techniques for recursive relational queries were developed for datalog

- But: no standard, no reference implementation; in HW4 we use Souffle
Outline

• Datalog rules

• Recursion

• Semantics

Next time: extensions, semi-naïve algo.
Datalog: Facts and Rules
Datalog: Facts and Rules

**Facts** = tuples in the database

**Rules** = queries

Actor(id, fname, lname)
Casts(pid, mid)
Movie(id, name, year)
Datalog: Facts and Rules

**Facts** = tuples in the database

**Rules** = queries

**Actor(id, fname, lname)**

**Casts(pid, mid)**

**Movie(id, name, year)**

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).
**Datalog: Facts and Rules**

**Facts** = tuples in the database

- `Actor(344759,'Douglas', 'Fowley').`
- `Casts(344759, 29851).`
- `Casts(355713, 29000).`
- `Movie(7909, 'A Night in Armour', 1910).`
- `Movie(29000, 'Arizona', 1940).`
- `Movie(29445, 'Ave Maria', 1940).`

**Rules** = queries

- `Q1(y) :- Movie(x,y,z), z='1940'.`
Datalog: Facts and Rules

Facts = tuples in the database

Actor(344759, 'Douglas', 'Fowley').
Casts(344759, 29851).
Casts(355713, 29000).
Movie(7909, 'A Night in Armour', 1910).
Movie(29000, 'Arizona', 1940).
Movie(29445, 'Ave Maria', 1940).

Rules = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Find Movies made in 1940
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

- Q1(y) :- Movie(x,y,z), z='1940'.
- Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).
Datalog: Facts and Rules

Facts = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Rules = queries

Q1(y) :- Movie(x,y,z), z=‘1940’.
Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).

Find Actors who acted in Movies made in 1940
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

- Q1(y) :- Movie(x, y, z), z='1940'.
- Q2(f, l) :- Actor(z, f, l), Casts(z, x), Movie(x, y, '1940').
- Q3(f, l) :- Actor(z, f, l), Casts(z, x1), Movie(x1, y1, 1910), Casts(z, x2), Movie(x2, y2, 1940).
Datalog: Facts and Rules

**Facts** = tuples in the database

- Actor(344759, ‘Douglas’, ‘Fowley’).
- Casts(344759, 29851).
- Casts(355713, 29000).

**Rules** = queries

- Q1(y) :- Movie(x,y,z), z=‘1940’.
- Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).
- Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910), Casts(z,x2), Movie(x2,y2,1940).

Find Actors who acted in a Movie in 1940 and in one in 1910
Datalog: Facts and Rules

Facts = tuples in the database

Actor(344759, ‘Douglas’, ‘Fowley’).
Casts(344759, 29851).
Casts(355713, 29000).
Movie(29445, ‘Ave Maria’, 1940).

Rules = queries

Q1(y) :- Movie(x,y,z), z=’1940’.
Q2(f, l) :- Actor(z,f,l), Casts(z,x),
              Movie(x,y,’1940’).
Q3(f,l) :- Actor(z,f,l), Casts(z,x1),
              Movie(x1,y1,1910),
              Casts(z,x2), Movie(x2,y2,1940)

Extensional Database Predicates = EDB = Actor, Casts, Movie
Intensional Database Predicates = IDB = Q1, Q2, Q3
Anatomy of a Rule

Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).
Anatomy of a Rule

Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'1940').
Anatomy of a Rule

Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).
Anatomy of a Rule

Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,’1940’).

f, l = head variables
x,y,z = existential variables
Outline

• Datalog rules

• Recursion

• Semantics

Next time: extensions, semi-naïve algo.
Datalog program

• A datalog program = several rules

• Rules may be recursive

• Set semantics only
Processing Graphs in Datalog

Graph

Pattern Matching

\[
R = \begin{array}{|c|c|}
\hline
\text{src} & \text{dst} \\
\hline
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
1 & 3 \\
\hline
\end{array}
\]
Processing Graphs in Datalog

Graph

Pattern Matching

\[ \text{Answer}(x, y, z) : R(x, y), R(x, z), R(y, z) \]
Example

Descendants of node 2

R encodes a graph

\[
\begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]
Example

Descendants of node 2

\[ D(x) :- R(2, x) \]
\[ D(y) :- D(x), R(x, y) \]

R encodes a graph

\[
\begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]
Example

R encodes a graph

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Descendants of node 2

D(x) :- R(2, x)
D(y) :- D(x), R(x, y)

Recursive rule
Example

R encodes a graph

Descendants of node 2

D(x) :- R(2, x)
D(y) :- D(x), R(x,y)

How recursion works in datalog:
Initially D = empty
Example

R encodes a graph

Descendants of node 2

\[ D(x) : \neg R(2, x) \]
\[ D(y) : \neg D(x), R(x,y) \]

How recursion works in datalog:
Initially \( D = \) empty
- Compute both rules:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
R encodes a graph

How recursion works in datalog:
Initially D = empty
  • Compute both rules:
    ...now D = {1,3}

Descendants of node 2

D(x) :- R(2, x)
D(y) :- D(x), R(x,y)

Recursive rule
R encodes a graph

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Descendants of node 2

D(x) :- R(2, x)
D(y) :- D(x), R(x, y)

How recursion works in datalog:
Initially D = empty
- Compute both rules:
  ...now D = \{1,3\}
- Compute both rules:
Example

R encodes a graph

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Descendants of node 2

D(x) :- R(2, x)
D(y) :- D(x), R(x,y)

How recursion works in datalog:
Initially D = empty
- Compute both rules:
  ...now D = \{1,3\}
- Compute both rules:
  ...now D = \{1,3,2,4\}
R encodes a graph

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Descendants of node 2

D(x) :- R(2, x)
D(y) :- D(x), R(x,y)

How recursion works in datalog:
Initially D = empty
- Compute both rules:
  ...now D = \{1,3\}
- Compute both rules:
  ...now D = \{1,3,2,4\}
- Compute both rules:
Example

R encodes a graph

Descendants of node 2

\[
D(x) : - R(2, x)
\]

\[
D(y) : - D(x), R(x, y)
\]

How recursion works in datalog:
Initially D = empty
- Compute both rules:
  ...now D = \{1,3\}
- Compute both rules:
  ...now D = \{1,3,2,4\}
- Compute both rules:
  ...now D = \{1,3,2,4,5\}

\[
R = \begin{array}{c|c}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5
\end{array}
\]
R encodes a graph

**Example**

Descendants of node 2

\[
D(x) :- R(2, x) \\
D(y) :- D(x), R(x, y)
\]

How recursion works in datalog:

- Initially \( D = \emptyset \)
- Compute both rules:
  - \( R(2, x) \)
  - \( D(x), R(x, y) \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( D = \{1, 3\} \)
- \( D = \{1, 3, 2, 4\} \)
- \( D = \{1, 3, 2, 4, 5\} \)
- \( D = \{2, 4, 1, 3, 5\} \)

STOP
Outline

• Datalog rules

• Recursion

• Semantics

Next time: extensions, semi-naïve algo.
Naïve Evaluation Algorithm

• Every rule $\rightarrow$ SPJ* query

*SPJ = select-project-join
*USPJ = union-select-project-join
Naïve Evaluation Algorithm

- Every rule $\rightarrow$ SPJ* query

$T(x,z) :- R(x,y), T(y,z), C(y,'green')$

*SPJ = select-project-join
*USPJ = union-select-project-join
Naïve Evaluation Algorithm

• Every rule $\rightarrow$ SPJ* query

\[ T(x,z) :: R(x,y), T(y,z), C(y,'green') \]

*SPJ = select-project-join
*USPJ = union-select-project-join
Naïve Evaluation Algorithm

- Every rule $\rightarrow$ SPJ* query

  $T(x,z) :- R(x,y), T(y,z), C(y,'green')$

- Multiple rules same head $\rightarrow$ USPJ+

  $T(x,y) :- \ldots$
  $T(x,y) :- \ldots$
  $\ldots$

$\ast$SPJ = select-project-join
$\ast$USPJ = union-select-project-join
Naïve Evaluation Algorithm

- Every rule $\rightarrow$ SPJ* query

\[
T(x,z) : R(x,y), T(y,z), C(y,'green')
\]

- Multiple rules same head $\rightarrow$ USPJ+

\[
T(x,y) : \ldots \quad \quad T(x,y) : \ldots \quad \ldots
\]

- Naïve Algorithm:

\[
IDBs := \emptyset \\
\textbf{repeat} \quad IDBs := USPJs \\
\textbf{until} \quad \text{no more change}
\]

*SPJ = select-project-join
*USPJ = union-select-project-join
Naïve Evaluation Algorithm

D(x) :- R(2,x)
D(y) :- D(x), R(x,y)
Naïve Evaluation Algorithm

\[
D(x) :- R(2,x)
\]

\[
D(y) :- D(x), R(x,y)
\]

\[
\Pi_{R.dst}(\sigma_{R.src=2}(R))
\]
Naïve Evaluation Algorithm

\[
\begin{align*}
D(x) &\space:\space R(2,x) \\
D(y) &\space:\space D(x), R(x,y)
\end{align*}
\]

\[
\Pi_{R.dst(\sigma_{R.src=2}(R))} \cup \Pi_{R.dst(D \bowtie_{D.node=R.src} R)};
\]
Naïve Evaluation Algorithm

\[ D(x) :- \ R(2,x) \]
\[ D(y) :- \ D(x), R(x,y) \]

\[ \Pi_{R.dst}(\sigma_{R.src=2}(R)) \cup \Pi_{R.dst}(D \bowtie_{D.node=R.src} R); \]
Naïve Evaluation Algorithm

\[ \begin{align*}
D(x) & \colon= R(2, x) \\
D(y) & \colon= D(x), R(x, y)
\end{align*} \]

\[
D := \emptyset; \\
\text{repeat} \\
\quad D := \Pi_{R.\text{dst}}(\sigma_{R.\text{src}=2}(R)) \cup \Pi_{R.\text{dst}}(D \bowtie_{D.\text{node}=R.\text{src}} R); \\
\text{until } [\text{no more change}]
\]
Naïve Evaluation Algorithm

The Naïve Evaluation Algorithm:

• Always terminates
• Always terminates in a number of steps that is polynomial in the size of the database