Elephants Can Split Graphs, or Very Large Graph Partitioning via PargreSQL

CONSTANTIN PAN AND MIKHAIL ZYMBLER,
SOUTH URAL STATE UNIVERSITY,
CHELYABINSK, RUSSIA
Outline

• PargreSQL DBMS in brief
• Graph partitioning via PargreSQL
• Experimental results
PargreSQL project

PostgreSQL + PARTITIONED PARALLELISM
Partitioned parallelism

\[ R_i = \{ t \mid t \in R, \phi(t) = i \} \]
i = 0, ..., 9

Fragmentation function
\[ \phi(t) = (t.id \div 10) \mod 10 \]
PostgreSQL vs PargreSQL
EXCHANGE operator
EXCHANGE operator

```
SELECT Name
FROM S, SP
WHERE S.S#=SP.S#
and
S.City='Heidelberg'
```
PargreSQL speedup

\[ R \bowtie S \]
\[ |R| = 6 \cdot 10^7 \]
\[ |S| = 1.5 \cdot 10^6 \]

\( \mu \) is a portion of tuples at every partition of the table to be sent to other nodes
Graph partitioning

Graph → Partitions

- cut size → min
- partition size ≈ partition size
Multilevel partitioning
Using PargreSQL

PargreSQL

```
coarsen  ...  coarsen
```

```
uncoarsen  ...  uncoarsen
```

Chaco

```
partition
```

6-Sep-13 DATABASE SYSTEMS RESEARCH GROUP SCIENTIFIC SEMINAR
Coarsening in memory

1. Find the heaviest (or a random) edge.

2. Collapse the edge into a vertex.

3. Merge the duplicates and remove the loops.

4. Repeat, avoiding the vertices generated this way, until nothing is left.
Coarsening with PargreSQL

1. Find the heaviest matching.
2. Collapse the edges of the matching into vertices.
3. Merge the duplicates and remove the loops.
Data flow
Coarsening implementation

**Graph**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

**Match**

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

**Coarse Graph**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>13</td>
</tr>
</tbody>
</table>
Uncoarsening implementation

Propagation

Gain calc.

Refining

<table>
<thead>
<tr>
<th>PARTITIONS</th>
<th>A</th>
<th>P</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 -2</td>
<td>2 1 -3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 0 -2</td>
<td>4 0 -1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PARTITIONS</th>
<th>A</th>
<th>P</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 -2</td>
<td>2 1 -3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 0 -2</td>
<td>4 0 -1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PARTITIONS</th>
<th>A</th>
<th>P</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 -2</td>
<td>2 1 -3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 0 -2</td>
<td>4 0 -1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Partitioning quality (by Kernighan and Lin)

\[
\text{gain}(v) = \sum_{(v,u) \in E, P(v) \neq P(u)} w(v,u) - \sum_{(v,u) \in E, P(v) = P(u)} w(v,u)
\]
Experiments

• Computer
  ◦ 128 nodes of Tornado cluster in South Ural State University (471st in top500)

• Data
  ◦ Luxembourg road map from OpenStreetMap (10^5 vertices, 1 iteration)
  ◦ Belgium road map from OpenStreetMap (10^6 vertices, 5 iterations)
  ◦ distributed over the cluster nodes by function \( \phi(e) = e.A \ast |V|/|E| \)
Time

- Luxembourg
- Belgium

Time, s

Computer nodes

10^5
10^4
10^3
10^2
10^1
10^0
10
1
8
16
32
64
128
Speedup

![Graph showing speedup with computer nodes and speed on the y-axis. The graph compares Luxembourg, Belgium, and linear performance.](image-url)
Coarsening and uncoarsening relative time

Belgium

[Bar chart showing coarsening and uncoarsening relative time for different computer node counts in Belgium]

Luxembourg

[Bar chart showing coarsening and uncoarsening relative time for different computer node counts in Luxembourg]
Random partitioning gives 30% miscolored vertices
Conclusion

• An approach to partition very large graphs by means of a relational parallel DBMS, that was implemented on the basis of PostgreSQL.

• Good speedup at an acceptable quality loss.

• Try different partitioning schemes and other very large graph problems in future.

• Papers describing this research were published in LNCS (DEXA 2013 and ADBIS 2013 proceedings).
Thanks for attention!

• Questions?
  ◦ Constantin Pan  kvapen@gmail.com
  ◦ Mikhail Zymbler  zymbler@gmail.com