ACM SIGMOD 2016 Programming Contest (uoa_team)
Athanasios-Michail Karampatsis, Nikos Dimakopoulos, Georgios Alexandropoulos, Nikos Tzamos
Yannis Foufoulas
Department of Informatics and Telecommunications, University of Athens, Greece
MaDgIK LAB, http://www.madgik.di.uoa.gr

Task description

- Calculate shortest path on a dynamic directed unweighted graph.
- An initial graph is given during the preprocessing stages.
- When the main execution starts, batches of queries are processed containing either updates or shortest path queries.
- Updates may concern insertions or deletions of edges.
- The results must be printed at the end of each batch.

Data Structures

- A hash map is used to create a dense adjacency list.
- A variation of an adjacency list that achieves higher cache locality is used to represent the whole graph.
- Two adjacency lists that keep the node names, along with their updates (insertions or deletions) and a version id are used to support multiversioning.

Preprocessing

- Load graph
- Mark as small, nodes with spanning trees that consist of less than 5 nodes.

Validation Processing

Find shortest path query?
Yes
Is node small?
Yes
Append insertion or deletion to the corresponding data structure and finish
No
Add query to concurrent queue that is being executed by the thread
No
Add query to queue that is being executed by the main thread

Use number of grandchildren as heuristic
Yes
Super nodes?
No
Use number of children as heuristic

Explore the smaller side

Updates?
Yes
multi versioning structure
No
Remove deleted edges

Visited from the other side?
Yes
Add new edges
No
Children?

Finish

Strategy

- Use an Enhanced Bidirectional BFS to calculate single-source shortest path.
- Parallelize shortest path queries to multiple threads using multiversioning data structures.
- Use heuristics to optimize multi-threading and Bidirectional BFS.

Graph Representation

<table>
<thead>
<tr>
<th>MAXINT table</th>
<th>Nodes</th>
<th>Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

Children of node 0
Children of node 1

Multithreading strategy

We addressed the following problems:

Skew
A major problem when running queries in parallel is skew. In order to eliminate it, a single concurrent queue is used and all threads consume jobs from this queue.

Locks
The concurrent queues are slower when there are many locks. This happens when there are many light queries and the threads request new jobs continuously.

We solved this problem by running light queries in single thread mode. Queries on small nodes are considered as light. Small nodes are marked during the preprocessing, and remain small if no insertions have been applied to their spanning trees.

Third party libraries used

- https://github.com/tghosgor/threadpool11 (LGPL v3.0)
- https://github.com/cameron314/concurrentqueue (Simplified BSD License)