Conquering LDBC SNB BI At SF-10K

15th LDBC TUC Meeting

Mingxi Wu, Yuchen Zhang

June 2022
Disclaimer

- This is not an LDBC benchmark result
- The result presented is ready for audit
- It should not be used as a baseline to be compared against
# LDBC Social Network Benchmark

<table>
<thead>
<tr>
<th></th>
<th>Interactive</th>
<th>Business Intelligence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Focus</strong></td>
<td>OLTP</td>
<td>OLAP</td>
</tr>
<tr>
<td><strong>Typical query</strong></td>
<td>2-3 hop neighborhood queries with filtering</td>
<td>Multi-hop/path/subgraph queries with filtering and aggregation</td>
</tr>
<tr>
<td><strong>Data size</strong></td>
<td>30G, 100G, 300G</td>
<td>100G, 300G, 1T, 3T, 10T, 36T</td>
</tr>
<tr>
<td><strong>Refresh operations</strong></td>
<td>Single-point inserts</td>
<td>Micro-batch inserts/deletes</td>
</tr>
<tr>
<td><strong>Target metric</strong></td>
<td>Throughput (ops/s)</td>
<td>Mean latency</td>
</tr>
</tbody>
</table>
BI Workload

- Target for OLAP
- Read Queries (touching a large portion of the data)
- Micro-batches of inserts and deletes
Setup - Hardware

- **Software**: TigerGraph 3.6.0 RC
- **CPU type**: GCP N2D CPU AMD Milan EPYC 7B12 2.25GHz
- **Operating System**: CentOS 7 v20220406
- **Disk type**: SSD persistent disk
- **Machine Configuration**:
  - SF-1K: 4 x n2d-highmem-64 (64 vCPU, 512 GB RAM, 2000 GB disk)
  - SF-10K: 30 x n2d-highmem-96 (96 vCPU, 768 GB RAM, 3200 GB disk)
Setup - Data

- **LDBC-SNB SF-10K**
- **Raw Initial data:** 11.64T
- **batch update data:**
  - Insert files: 1.59T
  - Delete files: 24G
- **Loading performance:**
  - Time: 5439s ~ 1.5h
  - Loading speed: ~256.8 GB/hr/machine
  - Disk size: 6.06T
### Results

<table>
<thead>
<tr>
<th>SF</th>
<th>load</th>
<th>w</th>
<th>r</th>
<th>q1</th>
<th>q2a</th>
<th>q2b</th>
<th>q3</th>
<th>q4</th>
<th>q5</th>
<th>q6</th>
<th>q7</th>
<th>q8a</th>
<th>q8b</th>
<th>q9</th>
<th>q10a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>4,295</td>
<td>1,758</td>
<td>6,199</td>
<td>136</td>
<td>127</td>
<td>28</td>
<td>315</td>
<td>51</td>
<td>18</td>
<td>18</td>
<td>42</td>
<td>36</td>
<td>16</td>
<td>522</td>
<td>293</td>
</tr>
<tr>
<td>10,000</td>
<td>5,439</td>
<td>4,165</td>
<td>16,954</td>
<td>190</td>
<td>268</td>
<td>114</td>
<td>974</td>
<td>88</td>
<td>65</td>
<td>71</td>
<td>213</td>
<td>99</td>
<td>39</td>
<td>1,112</td>
<td>684</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SF</th>
<th>q10b</th>
<th>q11</th>
<th>q12</th>
<th>q13</th>
<th>q14a</th>
<th>q14b</th>
<th>q15a</th>
<th>q15b</th>
<th>q16a</th>
<th>q16b</th>
<th>q17</th>
<th>q18</th>
<th>q19a</th>
<th>q19b</th>
<th>q20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>70</td>
<td>89</td>
<td>182</td>
<td>755</td>
<td>484</td>
<td>70</td>
<td>762</td>
<td>1,422</td>
<td>101</td>
<td>31</td>
<td>117</td>
<td>333</td>
<td>81</td>
<td>78</td>
<td>22</td>
</tr>
<tr>
<td>10,000</td>
<td>119</td>
<td>181</td>
<td>442</td>
<td>2,266</td>
<td>1,506</td>
<td>324</td>
<td>2,094</td>
<td>3,195</td>
<td>433</td>
<td>78</td>
<td>319</td>
<td>1,602</td>
<td>217</td>
<td>223</td>
<td>41</td>
</tr>
</tbody>
</table>

- **w** (write): sum of insert, delete and precompute time
- **r** (read): sum of read queries q1 ~ q20 time (in total 27 variations)
- **qi**: sum of 10 runs time of different input parameters
- **Overall**: batch = w + r
  - SF-1K: 1 batch takes 7,957s = 2.21h, (23s/query, total 27 query)
  - SF-10K: 1 batch takes 21,119s = 5.8h, (63s/query, total 27 query)
Implementation - Precomputation

Precomputation is used in BI 4, 6, 19 and 20

For example, query 4:

1. Precomputes: for a given forum, calculate the max MemberCount of all countries and store it as an auxiliary attribute of the forum
2. When querying, get the top 100 forums using the auxiliary attribute
Implementation - shortest paths

Query 15, 19, 20 is related to shortest path searching

1. Precomputes: the edge weight is calculated and inserted as weighted edge
2. When querying, we used GSQL query to search the shortest paths on the added edges.

Query 20
Implementation - insert

**Insert operations** are performed using GSQL loading scripts

- Files are distributed on cluster and is loaded concurrently
- Same speed as the initial loading job
Delete operations are cascade. Deletes are done by GSQL DML queries.

- The vertices/edges are selected in the query
- The selected vertices and edges are deleted