Graph Storage Considerations

Andreas Schmidt

Department of Informatics and Business Information Systems
University of Applied Sciences Karlsruhe
Germany

Institute for Applied Computer Sciences
Karlsruhe Institute of Technology
Germany
Observation

The time to answer a query is mainly determined by the amount of data that has to be loaded from disk / from main memory.
Example Szenario

Person 3 is friend of Person 6
Person 6 is friend of Person 7
Person 7 is friend of Person 8
Person 8 is friend of Person 9
Person 9 is friend of Person 10
Person 10 is friend of Person 11
Person 11 is friend of Person 12
Person 12 is friend of Person 13
Person 13 is friend of Person 4

Andreas Schmidt - GraphSM - 2016
Example Query

- Example Query: Show all my friends of grade n

- Dataset 1: 1000 users, on average 50 friends/person

<table>
<thead>
<tr>
<th>grade</th>
<th>execution time MySQL (sec.)</th>
<th>execution time neo4j db (sec.)</th>
<th>number results</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.028</td>
<td>0.04</td>
<td>~900</td>
</tr>
<tr>
<td>3</td>
<td>0.213</td>
<td>0.06</td>
<td>999</td>
</tr>
<tr>
<td>4</td>
<td>10.273</td>
<td>0.07</td>
<td>999</td>
</tr>
<tr>
<td>5</td>
<td>92.613</td>
<td>0.07</td>
<td>999</td>
</tr>
</tbody>
</table>

Source: Neo4j in Action, Manning Verlag, 2015
Example Query

- Example Query: Show all my friends of grade n

- Dataset 2: 1,000,000 users, on average 50 friends/person

<table>
<thead>
<tr>
<th>grade</th>
<th>query time MySQL (sec.)</th>
<th>query time neo4j (sec.)</th>
<th>number results</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.016</td>
<td>0.010</td>
<td>~2500</td>
</tr>
<tr>
<td>3</td>
<td>30.267</td>
<td>0.168</td>
<td>~125,000</td>
</tr>
<tr>
<td>4</td>
<td>1,543.505</td>
<td>1.359</td>
<td>~600,000</td>
</tr>
<tr>
<td>5</td>
<td>not finished</td>
<td>2.132</td>
<td>999.999</td>
</tr>
</tbody>
</table>

Source: Neo4j in Action, Manning Verlag, 2015
Relational Database - Data Representation?

- Table `is_friend_of`
  - | person | friend |
  - | 3      | 6      |
  - | 3      | 7      |
  - | 3      | 11     |
  - | 6      | 3      |
  - | 6      | 8      |
  - | 6      | 11     |
  - | 6      | 17     |
  - | 7      | 6      |
  - | 7      | 12     |
  - | ...    | ...    |

- Index `is_friend_of(person, friend)`
  - Complexity: $O(\log(n))$

- Diagram showing an index with all persons with id in [300, 688] highlighted.
Graph Database - Data Representation (extract)

- Nodes (fixed record length):
  01: node 1:
  02: node 2:
  03: node 3: 1-3, ...
  04: node 4:
  05: node 5:
  06: node 6: 4-7, ...
  07: node 7: 8-9, ...
  08: 
  09: 
  10: 
  11: 

- Relations (fixed record length):
  01: 3 --> 6, is_friend_of, -, 2
  02: 3 --> 7, is_friend_of, 1, 3
  03: 3 --> 11, is_friend_of, 2, -
  04: 6 --> 3, is_friend_of, -, 5
  05: 6 --> 8, is_friend_of, 4, 6
  06: 6 --> 11, is_friend_of, 5, 7
  07: 6 --> 17, is_friend_of, 6, -
  08: 7 --> 6, is_friend_of, -, 9
  09: 7 --> 12, is_friend_of, 8, -
  10: 
  11: 

Complexity: O(1)
Other Example:

- Query: Return all pairs of persons who have at least 2 common friends, but did not know each other
- Problem with this query?
## Numbers everyone should know (from [Lad09])

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>100</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>10,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from network</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>30,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA 1</td>
<td>50,000,000</td>
</tr>
</tbody>
</table>

### Figure 3

**Comparing Random and Sequential Access in Disk and Memory**

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random, disk</td>
<td>316 values/sec</td>
</tr>
<tr>
<td>Sequential, disk</td>
<td>53.2M values/sec</td>
</tr>
<tr>
<td>Random, SSD</td>
<td>1924 values/sec</td>
</tr>
<tr>
<td>Sequential, SSD</td>
<td>42.2M values/sec</td>
</tr>
<tr>
<td>Random, memory</td>
<td>36.7M values/sec</td>
</tr>
<tr>
<td>Sequential, memory</td>
<td>358.2M values/sec</td>
</tr>
</tbody>
</table>

*Note: Disk tests were carried out on a freshly boot ed machine (a Windows 2003 server with 64-GB RAM and eight 15,000-RPM SAS disks in RAID5 configuration) to eliminate the effect of operating-system disk caching. SSD test used a latest-generation Intel high-performance SATA SSD.*

Cache & Cacheline

memory request (from CPU) → load into CPU → another memory request

steps (1)...(4): cache miss
steps (5) + (6): cache hit

load into CPU

load into CPU

load into CPU

load into CPU
Resumee

- Native Graph databases support index-free adjacency
- Choose the right database type, according for your application demands
- Knowing the underlying storage model can help to chose the right database type
- Data from the first level cache can be accessed 200 times faster, compared to data that resides in main memory (cache consciousness)
- Sequential access is between one and five orders of magnitude faster than random access
that’s all ....

thanks