Horton: online query execution on large distributed graphs

Sameh Elnikety
Microsoft Research
People

• Team

  - Alan Geller
  - Jim Larus
  - Sergey Bykov
  - CJ Williams
  - Jorgen Thelin
  - Suyash Sinha
  - Gabriel Kliot
  - Ravi Pandya
  - Yuxiong He

• Interns
  - Mohamed Sarwat (UMN)
  - Jiaqing Du (EPFL)
Large Graphs Are Important

- Large graphs appear in many application domains
  - Examples: social networks, knowledge representation, …
Example: Social Network

- **Scale**
  - LinkedIn
    » 70 million users
  - Facebook
    » 500 million users
    » 65 Billion photos

- **Queries**
  - Find Alice’s friends
  - Find Alice’s photos with friends

- **Rich graph**
  - Types, attributes
Objective: Management & Querying

• Manage and query large graphs online
  – Today
    » Expensive hardware (limited)
    » Cluster of machines (ad-hoc)
  – Tomorrow
    » Growth area

• Analogy: provide system support
  » Analogy: DBMS manages application data
Key Design Decisions

• Interactive queries
  – Graph stored in random access memory

• Graph too large for one machine
  – Distributed problem
  – Partitioning: slice graph into pieces
  – Replication: for fault tolerance

• Long time to build graph
  – Support updates
Three Research Problems

1. How to partition the graph
   - Static and dynamic techniques
     » E.g., Evolving set partitioning

2. How to query the graph
   - Data model & query language
   - Execution engine
   - Query optimizations

3. How to update the graph
   - Multi-version concurrency control
     » E.g., Generalized Snapshot Isolation
Concurrent Control

- Generalized Snapshot Isolation – **GSI**
- Replicas agree
  - On which update transactions commit
  - On their commit order
- Example (bank database)
  - T1: set balance = $1000
  - T2: set balance = $2000
  - Replica1: see T1 then T2 → balance = $2000
  - Replica2: see T2 then T1 → balance = $1000
Replication

Load Balancer

Single Graph Server
Replica 2
Replica 3
Read Tx

Load Balancer → Replica 1
→ Replica 2
→ Replica 3
Update Tx
Replication Middleware

Replica 1
A → B → C

durability OFF

Replica 2
A → B → C

durability OFF

Replica 3
A → B → C

durability OFF

Replication MW (global ordering)
A → B → C

durability
A → B → C

Tx A

Tx B

Tx C
Partitioning – Key Ideas

1. Objective
   - Maximize temporal and spatial locality

2. Static
   - Linear time
     » Evolving-set partitioning
   - Streaming algorithm
     » During loading

3. Dynamic
   - Incremental
     » Affinity propagation
Three Research Problems

1. How to partition the graph
   - Static and dynamic techniques
     » E.g., Evolving set partitioning

2. How to query the graph
   - Data model & query language
   - Execution engine
   - Query optimizations

3. How to update the graph
   - Multi-version concurrency control
     » E.g., Generalized Snapshot Isolation
Data Model

- **Node**
  - ID, type, attributes

- **Edge**
  - Connects two nodes
  - Direction, type, attributes

**App**

![App Diagram]

**Horton**

![Horton Diagram]
Query Language

• Trade-off
  – Expressiveness vs. efficiency

• Regular language reachability
  – Query is a regular expression
    » Sequence of node and edge predicates
  – Example
    » Alice’s photos
    » Photo, tags, Alice
    » Node: type=photo, edge: type=tags, node: type=person, name=Alice
    » Result: matching paths
Query Language - Operators

• Projection
  » Alice’s photos
  » Photo, tags, Alice
  » **Node: type=photo**, **edge: type=tags**, **node: type=person, name=Alice**
  » **SELECT** photo
     **FROM** photo, tags, Alice

• OR
  » (Photo | video), tags, Alice

• Kleene star
  » Alice org chart
  » **Alice, (manages, person)***
Example App: CodeBook - Queries

1. Person, FileOwner>, TFSFile, FileOwner<, Person

2. Person, DiscussionOwner>, Discussion, DiscussionOwner<, Person

3. Person, WorkItemOwner>, TFSWorkItem, WorkItemOwner< ,Person

4. Person, Manages<, Person, Manages>, Person

5. Person, WorkItemOwner>, TFSWorkItem, Mentions>, TFSFile, Mentions>, TFSWorkItem, WorkItemOwner<, Person

6. Person, WorkItemOwner>, TFSWorkItem, Mentions>, TFSFile, FileOwner<, Person

7. Person, FileOwner>, TFSFile, Mentions>, TFSWorkItem, Mentions>, TFSFile, FileOwner<, Person
Query Language - Future Extensions

- **From**: path
  - Sequence of predicates

- **To**: sub-graph
  - Graph pattern

- Sub-graph isomorphism
Talk Outline

• Graph query language
• **Graph query execution engine**
• Graph query optimizer
• Initial experimental results
• Demo
Execution Engine

1. Build a FSM from input query
2. Optimize FSM
3. Executes FSM using distributed BFS
Centralized Query Execution

Answer Paths: Alice, Tags, Photo
Alice, Tags, Photo8
Distributed Execution Engine

1. Query to Convert Query To FSM
2. Send FSM to needed Partitions
3. Partitions 1, 2, ..., N
   - Asynchronous Communication
   - Query Execution
4. Streaming
5. Results
Distributed Query Execution

Alice, Tags, Photo, Tags, Hillary
Distributed Query Execution

Alice, Tags, Photo, Tags, Hillary

Step 1
Partition 1
Alice
Partition 2
Photo1 Photo8

Step 2
Partition 1
Hillary
Partition 2

Step 3
FSM

Partition 1
Alice Tags
S0
S1
S2
S3
S4
S5

Partition 2
Bob Chris David

Hillary
Ed France George
# Graph Query Optimization

<table>
<thead>
<tr>
<th>Query</th>
<th>Execution Time without Optimization</th>
<th>Execution Time with Optimization</th>
<th>Result size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person, activated &gt;, TFSWorkItemRevision, Links &gt;, WebURL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>remote edges = 4643</td>
<td>remote edges = 7</td>
<td>230 msec (R to L)</td>
</tr>
<tr>
<td></td>
<td>messages = 33</td>
<td>messages = 7</td>
<td></td>
</tr>
<tr>
<td>Person, Manages &gt;, Person, Closed &gt;, TFSWorkItemRevision</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>remote edges = 72969</td>
<td>remote edges = 41</td>
<td>700 msec (R to L)</td>
</tr>
<tr>
<td></td>
<td>messages = 45</td>
<td>messages = 24</td>
<td></td>
</tr>
<tr>
<td>Person, Manages &gt;, Person 134398185, Committer &gt;, TFSCheckin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>remote edges = 73045</td>
<td>remote edges = 107</td>
<td>390 msec (split at 2, 0-2 R to L, 2-4 L to R)</td>
</tr>
<tr>
<td></td>
<td>messages = 35</td>
<td>messages = 6</td>
<td></td>
</tr>
<tr>
<td>Person, Manages &gt;, Person, Committer &gt;, TFSCheckin 671112371</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>remote edges = 73045</td>
<td>remote edges = 107</td>
<td>200 msec (R to L)</td>
</tr>
<tr>
<td></td>
<td>messages = 35</td>
<td>messages = 6</td>
<td></td>
</tr>
</tbody>
</table>
Graph Query Optimization

• Input
  – Graph statistics + query plan
• Output
  – Optimized query plan (execution sequence)
• Action
  – Enumerate execution plans
  – Evaluate their costs using graph statistics
  – Find the plan with minimum cost
Graph Query Optimization Techniques

1. Predicate ordering
2. Derived edges
3. Fragment reuse
Predicate Ordering

Find Mike’s photo that is also tagged by at least one of his friends

Different predicate orders can result in different execution costs.

Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike
Predicate Ordering

Find Mike’s photo that is also tagged by at least one of his friends

Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

Execute left to right
Predicate Ordering

Find Mike’s photo that is also tagged by at least one of his friends

Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

Execute left to right
Predicate Ordering

Find Mike’s photo that is also tagged by at least one of his friends

Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

Execute left to right
Predicate Ordering

Find Mike’s photo that is also tagged by at least one of his friends

Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

Total cost = 14
Predicate Ordering

Find Mike’s photo that is also tagged by at least one of his friends

Different predicate orders can result in different execution costs.

Execute left to right: Total cost = 14

Execute right to left: Total cost = 7
How to Decide Predicate Ordering?

• Enumerate execution sequences of predicates
• Estimate their costs using graph statistics
• Find the sequence with minimum cost
Cost Estimation using Graph Statistics

Graph Statistics

<table>
<thead>
<tr>
<th>Node type</th>
<th>#nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>5</td>
</tr>
<tr>
<td>Photo</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>FriendOf</th>
<th>Tagged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>1.2</td>
<td>2.2</td>
</tr>
<tr>
<td>Photo</td>
<td>N/A</td>
<td>1.6</td>
</tr>
</tbody>
</table>

**Left to right**
Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

EstimatedCost = ???
Cost Estimation using Graph Statistics

Graph Statistics

<table>
<thead>
<tr>
<th>Node type</th>
<th>#nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>5</td>
</tr>
<tr>
<td>Photo</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node type</th>
<th>FriendOf</th>
<th>Tagged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>1.2</td>
<td>2.2</td>
</tr>
<tr>
<td>Photo</td>
<td>N/A</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Left to right
Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

EstimatedCost = 1 [find Mike]
Cost Estimation using Graph Statistics

Graph Statistics

<table>
<thead>
<tr>
<th>Node type</th>
<th>#nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>5</td>
</tr>
<tr>
<td>Photo</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node type</th>
<th>FriendOf</th>
<th>Tagged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>1.2</td>
<td>2.2</td>
</tr>
<tr>
<td>Photo</td>
<td>N/A</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Estimated Cost = 1 + (1 * 2.2)
[find Mike] [find Mike, Tagged, Photo]
Cost Estimation using Graph Statistics

Graph Statistics

<table>
<thead>
<tr>
<th>Node type</th>
<th>#nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>5</td>
</tr>
<tr>
<td>Photo</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node type</th>
<th>FriendOf</th>
<th>Tagged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>1.2</td>
<td>2.2</td>
</tr>
<tr>
<td>Photo</td>
<td>N/A</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Left to right
Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

EstimatedCost = 1 [find Mike] + (1 * 2.2) [find Mike, Tagged, Photo] + (2.2 * 1.6) [find Mike, Tagged, Photo, Tagged, Person] + (2.2 * 1.6 * 1.2) [find Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike] = 11
Plan Enumeration

Find Mike’s photo that is also tagged by at least one of his friends

Plan1
Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

Plan2
Mike, FriendOf, Person, Tagged, Photo, Tagged, Mike

Plan3
(Mike, FriendOf, Person) ▶ (Person, Tagged, Photo, Tagged, Mike)

Plan4
(Mike, FriendOf, Person, Tagged, Photo) ▶ (Photo, Tagged, Mike)
Derived Edges

Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike  Cost = 7

Mike, Tagged, Photo, Friends-photo-tag, Mike  Cost = 5
Query Fragment Reuse

**Query1**
Mike, Tagged, Photo

**Query2**
Person, FriendOf, Mike

**Query3**
Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike
Query Fragment Reuse

**Query1**
Mike, Tagged, Photo

**Query2**
Person, FriendOf, Mike

**Query3**
Mike, Tagged, Photo, Tagged, Person, FriendOf, Mike

Execute Query3 based on cached results of Query1 and Query2:

[Query1], Tagged, Person, FriendOf, Mike

[Query2], Tagged, Photo, Tagged, Mike

[Query1] ⚤ [Photo, Tagged, Person] ⚤ [Query2]
Summary of Query Optimization

- Predicate ordering
- Derived edges
- Reuse of query fragments
Experimental Evaluation

Graph
- Codebook application
- 1 Million nodes
- 10 million edges

Machines
- 7 machines
- Intel Core 2 Duo 2.26 GHz , 4 GB ram
- Graph partitioned on 6 machines
## Initial Results

<table>
<thead>
<tr>
<th>Query</th>
<th>Execution Time without Optimization</th>
<th>Execution Time with Optimization</th>
<th>Result size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person, activated &gt;, TFSWorkItemRevision, Links &gt;, WebURL</td>
<td>7 sec</td>
<td>230 msec (R to L)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>remote edges = 4643 messages = 33</td>
<td>remote edges = 7 messages = 7</td>
<td></td>
</tr>
<tr>
<td>Person, Manages &gt;, Person, Closed &gt;, TFSWorkItemRevision</td>
<td>36 sec</td>
<td>700 msec (R to L)</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>remote edges = 72969 messages = 45</td>
<td>remote edges = 41 messages = 24</td>
<td></td>
</tr>
<tr>
<td>Person, Manages &gt;, Person 134398185, Committer &gt;, TFSCheckIn</td>
<td>32 sec</td>
<td>390 msec (split at 2, 0-2 R to L, 2-4 L to R)</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>remote edges = 73045 messages = 35</td>
<td>remote edges = 107 messages = 6</td>
<td></td>
</tr>
<tr>
<td>Person, Manages &gt;, Person, Committer &gt;, TFSCheckin 671112371</td>
<td>1 min</td>
<td>200 msec (R to L)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>remote edges = 73045 messages = 35</td>
<td>remote edges = 107 messages = 6</td>
<td></td>
</tr>
</tbody>
</table>
Summary

• Graph query language
  – Regular language expression
• Graph query execution engine
  – Distributed breadth first-search
• Graph query optimizer
  – Predicate ordering, derived edges, query fragment reuse
• Implementation
  – Initial experimental results
  – Demo