SociaLite: A Datalog-based Language for Large-Scale Graph Analysis

Jiwon Seo

MOBI SOCIAL RESEARCH GROUP
Overview
Overview

- SociaLite: language for large-scale graph analysis
- Extensions to Datalog
- Compiler optimizations for SociaLite queries
- Graph algorithms in SociaLite

- HW 6 (using SociaLite)
Motivation & Challenges

- Analysis of large-scale graphs are important
  - IT Industry – Twitter, LinkedIn, FaceBook, Pinterest
  - Bio-informatics, etc

- Challenges
  - Difficulty of distributed programming
  - Complexity of graph algorithms
State of the Art Technology

- MapReduce
  - MapReduce based graph systems (HaLoop)
- Pregel
  - vertex-centric programming

→ Too low-level programming model
Distributed Graph Language

- SociaLite
  - Abstractions for graph algorithms
  - Datalog-based query language
  - Graph algorithms in high-level language, and compiled to parallel/distributed code
Single-Source Shortest Paths (SSSP)

- Shortest distances from a single source node to rest of the nodes in a graph
- Core graph algorithm & a running example in this talk
SSSP in Datalog

- Two tables to store the graph and path distances
  - Edge(src, target, length).
  - Path(target, distance).
  - MinPath(target, minimum-distance).

\[
\begin{align*}
\text{Path}(t, d) & : \text{Edge}(1, t, d). \quad (1) \\
\text{Path}(t, d) & : \text{Path}(s, d_1), \text{Edge}(s, t, d_2), d = d_1 + d_2. \quad (2) \\
\text{MinPath}(t, \min(d)) & : \text{Path}(t, d). \quad (3)
\end{align*}
\]
## Datalog Performance

### Execution times for SSSP*

<table>
<thead>
<tr>
<th></th>
<th>Exec Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlog</td>
<td>24.9</td>
</tr>
<tr>
<td>IRIS</td>
<td>12.0</td>
</tr>
<tr>
<td>LogicBlox</td>
<td>3.4</td>
</tr>
<tr>
<td>Java**</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- Datalog 30~250 times slower

* synthetic graph with 100K nodes, 1M edges
** Java program implemented in Dijkstra’s algorithm
Why is Datalog Slow?

- Execution time wasted in sub-optimal paths

\[
\begin{align*}
\text{Path}(t, d) & : - \text{Edge}(1, t, d). \\
\text{Path}(t, d) & : - \text{Path}(s, d_1), \text{Edge}(s, t, d_2), d=d_1+d_2. \\
\text{MinPath}(t, \$\min(d)) & : - \text{Path}(t,d).
\end{align*}
\]
Why is Datalog Slow?

- Inefficient data structure for graphs

Java, adjacency list vs Datalog, flat table
Extensions in SociaLite

- Recursive aggregate functions
- Tail-nested tables (data layout extension)
Recursive Aggregate Functions

- Aggregate functions inside recursion
- Pruning suboptimal answers

- Syntax and Semantics

Path(t, $\min(d))$ :- Edge(1,t,d) ;
  :- Path(s, d₁), Edge(s, t, d₂), d=d₁+d₂.

Shortest Paths in SociaLite
Fixed-Point Semantics

- Equation for recursive rules:
  \[ R = h(R) = g \circ f(R) \]

  e.g. \[ f(R) = \{ \langle t, d \rangle | \text{Edge}(1, t, d) \lor \langle s, d_1 \rangle \in R \land \text{Edge}(s, t, d_2) \land d = d_1 + d_2 \} \]
  \[ g(R) = \{ \langle t, \min_{(t, d) \in R} d \rangle \} \]

  \text{Path}(t, \min(d)) :- \text{Edge}(1, t, d) ;
  \text{Path}(s, d_1), \text{Edge}(s, t, d_2), d = d_1 + d_2.
R = h(R) = g \circ f (R)

- If $g$ is a meet operator, and $f$ is monotone under $g$
- naïve evaluation converges to a fixed-point
- the solution is *greatest fixed-point solution*

- $g$ induces partial order $\leq$ and semi-lattice
- $x \leq y \Rightarrow f(x) \leq f(y)$ where $\leq$ is from $g$
Fixed-Point Semantics

\[ R = h(R) = g \circ f(R) \]

- If \( g \) is a meet operator, and \( f \) is monotone under \( g \)
  - naïve evaluation converges to a fixed-point
  - the solution is *greatest fixed-point solution*

- \( g \) induces partial order \( \subseteq \) and semi-lattice
- \( x \subseteq y \Rightarrow f(x) \subseteq f(y) \) where \( \subseteq \) is from \( g \)

Path(t, \$\text{min}(d)) :\neg \text{Edge}(1,t,d); \neg \text{Path}(s, d_1), \text{Edge}(s, t, d_2), d=d_1+d_2.$
Optimizations

- Semi-naïve evaluation
  - Optimized evaluation for recursion
  - Uses delta (new tuples) as input for evaluation
  - Transforms shortest-paths to Bellman-Ford

- Prioritization by $\leq$ (partial order by $g$)
  - e.g. in shortest-paths, prioritize the tuple with minimum distance ($\min$)
  - Transforms the program into Dijkstra’s algorithm

* SociaLite: Datalog Extensions for Efficient Social Network Analysis, ICDE’13
Tail-nested Tables

- Tables with nesting at the last column
- Annotation in table declarations
  
  e.g. \texttt{Edge(int s, (int t))}.

- Data as index
  
  - Range annotation \((N1..N2)\) \(\rightarrow\) array index
  
  e.g. \texttt{Edge(int s:1..10, (int t))}. 

Edge(int s, int t).

Edge(int s, (int t)).

Edge(int s:0..10, (int t)).
Distributed Tables

- Partitioning (sharding) by first column
- Hash-based partitioning

Edge(int src, int target)).
Distributed Tables

- Range-based partitioning

```
Edge(int src: 0..10, (int target)).
```
Distributed Execution

Foo(int a, int b).
Bar(int a, int b).
Qux(int a, int b).

Foo(a, c) :- Bar(a, b), Qux(b, c).
Parallel Prioritization of Recursive Aggregation

\[ \text{Path}(t, \min(d)) : - t=0, d=0; \]
\[ : - \text{Path}(s, d_1), \text{Edge}(s, t, d_2), d=d_1+d_2. \]

\[ \Delta \text{Path} = \begin{cases} 
\text{Path}(1, 1.1) \\ 
\text{Path}(2, 1.5) \\ 
\text{Path}(1, 1.2) \\ 
(d < 2.0) 
\end{cases} \quad \begin{cases} 
\text{Path}(3, 2.1) \\ 
\text{Path}(4, 2.7) \\ 
\text{Path}(0, 2.3) \\ 
(2.0 < d < 3.0) 
\end{cases} \quad \begin{cases} 
\text{Path}(9, 3.1) \\ 
\text{Path}(7, 3.9) \\ 
\text{Path}(4, 3.3) \\ 
(3.0 < d < 4.0) 
\end{cases} \quad \cdots \]

Δ Path = (d < 2.0) \subseteq (2.0 < d < 3.0) \subseteq (3.0 < d < 4.0)

→ Delta-stepping algorithm generalized, applied to recursive aggregate functions
Other Optimizations

- Condition Folding
  - Binary search for sorted columns
  e.g. \( \text{Foo}(a, b) :- \text{Bar}(a, b), b > 10. \)

- Pipelining
  - Evaluate multiples rules at once (locality)
  e.g. \( \text{Foo}(a, b) :- \text{Bar}(a, b), b > 10. \)
  \( \text{Baz}(a, b) :- \text{Foo}(a, b), c=b*b. \)
Other Optimizations

Approximate Computation

- Table columns as Bloom Filter
- Store large intermediate results approximately
Foaf(i, ff) :- Friend(i, f), Friend(f, ff).
LocalCount(i, $inc(1)) :- Foaf(i, ff), Attr(ff, “Some Attr”).
Approximation w/ Bloom Filter

Foaf(i, ff) :- Friend(i, f), Friend(f, ff).
LocalCount(i, $inc(1)) :- Foaf(i, ff), Attr(ff, “Some Attr”).

(2nd column of Foaf table is represented with a Bloom filter)
Approximation w/ Bloom Filter

Foaf(i, ff) :- Friend(i, f), Friend(f, ff).
LocalCount(i, $inc(1)) :- Foaf(i, ff), Attr(ff, "Some Attr").

(2nd column of Foaf table is represented with a Bloom filter)

<table>
<thead>
<tr>
<th></th>
<th>Exact</th>
<th>Approximation</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exec time (min)</td>
<td>28.9</td>
<td>19.4</td>
<td>32.8% faster</td>
</tr>
<tr>
<td>Memory usage(GB)</td>
<td>26.0</td>
<td>3.0</td>
<td>11.5% usage</td>
</tr>
<tr>
<td>Accuracy(&lt;10% error)</td>
<td>100.0%</td>
<td>92.5%</td>
<td></td>
</tr>
</tbody>
</table>

* LiveJournal (4.8M nodes, 68M edges)
**Python Integration**

- SociaLite queries embedded in Python code
  - `Queries are quoted in backtick`
- Python ↔ SociaLite
  - Python functions, variables are accessible in SociaLite queries (with prefix $)
  - SociaLite tables are readable from Python
Python Integration

PageRank in Python & SociaLite

```
Rank(n, 0, r) :- Node(n), r=1.0/$N.```

```python
for i in range(50):
  `Rank(p_i, $i+1, $sum(r)) :- Node(p_i), r=0.15*1.0/$N;
    :- Rank(p_j, $i, r_1), Edge(p_j, p_i),
        EdgeCnt(p_j, cnt), r=0.85*r_1/cnt.`
```
Benchmark algorithms

- Shortest-Paths
- PageRank
- Mutual Neighbors
- Connected Components
- Finding Triangles
- Clustering Coefficients

→ Evaluated on single-core multi-core/distributed cluster
## Input Graph for Single-Core

<table>
<thead>
<tr>
<th>Source</th>
<th>Type</th>
<th>Size</th>
<th>Used for</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiveJournal</td>
<td>directed</td>
<td>4.8M nodes 68M edges</td>
<td>Shortest Paths PageRank</td>
<td>Intel Xeon 2.80GHz 32GB memory</td>
</tr>
<tr>
<td>Last.fm</td>
<td>undirected</td>
<td>1.7M nodes 6.4M edges</td>
<td>Mutual Neighbors Connected Components Triangle Clustering Coefficients</td>
<td>Intel Core2 2.66GHz 3GB memory</td>
</tr>
</tbody>
</table>
Speedup from Tail-nested Tables

- Shortest Paths
- PageRank
- Mutual Neighbors
- Connected Components
- Triangles
- Clustering Coefficients
Speedup from Other Optimizations

- **Condition Folding**
- **Pipelining**
- **Prioritization**
- **Nested table**
- **Baseline**

The chart shows the speed-up for various optimization techniques across different benchmarks:

- **Shortest Paths**
- **PageRank**
- **Mutual Neighbors**
- **Connected Components**
- **Triangles**
- **Clustering Coefficients**
SociaLite (w/ optimizations)

- Shortest Paths
- PageRank
- Mutual Neighbors
- Connected Components
- Triangles
- Clustering Coefficients

speedup over Java
Optimized Java vs SociaLite

- Shortest Paths
- PageRank
- Mutual Neighbors
- Connected Components
- Triangles
- Clustering Coefficients

Opt Java
SociaLite

speedup over initial Java
### Input Graph for Multi-Core

<table>
<thead>
<tr>
<th>Source</th>
<th>Size</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friendster</td>
<td>120M nodes 2.5B edges</td>
<td>Intel Xeon E5-2670 16 cores(8+8) 2.60GHz 20MB last-level cache 256GB memory</td>
</tr>
</tbody>
</table>
Parallel Performance (Multi-Core)

**Shortest Paths**
- Execution Time vs. Number of Cores
- Parallelization Speedup vs. Number of Cores

**PageRank**
- Execution Time vs. Number of Cores
- Parallelization Speedup vs. Number of Cores

**Mutual Neighbors**
- Execution Time vs. Number of Cores
- Parallelization Speedup vs. Number of Cores

**Connected Components**
- Execution Time vs. Number of Cores
- Parallelization Speedup vs. Number of Cores

**Triangle**
- Execution Time vs. Number of Cores
- Parallelization Speedup vs. Number of Cores

**Clustering Coefficients**
- Execution Time vs. Number of Cores
- Parallelization Speedup vs. Number of Cores
## Input Graph for Distributed Evaluation

<table>
<thead>
<tr>
<th>Source</th>
<th>Size</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic Graph*</td>
<td>up to 268M nodes 4.3B edges (weak scaling)</td>
<td>64 Amazon EC2 Instances Intel Xeon X5570, 8 cores 23GB memory</td>
</tr>
</tbody>
</table>

*RMAT algorithm, Graph 500 Generator*
Distributed Performance

- Shortest Paths
- PageRank
- Mutual Neighbors
- Connected Components
- Triangle
- Clustering Coefficients

Exec Time (Sec.)

- Socialite
- Ideal (BF)
- Ideal

# of machines

Exec Time (Min.)
Giraph (Pregel) vs SociaLite

- **Exec Time (Sec.)**: Shortest paths, PageRank, Mutual neighbors, Connected Components, Triangle, Clustering Coefficients
- **Exec Time (Min.)**: Shortest paths, PageRank, Mutual neighbors, Connected Components, Triangle, Clustering Coefficients
- **# of machines**: 2, 4, 8, 16, 32, 64
Pregel (Giraph)

- Programming model:
  - Vertex-centric model
  - (manual) message passing

- Implement function F
  - Executed at each iteration
  - Process messages received
  - Send out messages (delivered next iteration)
## Programmability (Distributed)

- Giraph (Pregel) vs SociaLite (lines of code)

<table>
<thead>
<tr>
<th></th>
<th>Giraph</th>
<th>SociaLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest Paths</td>
<td>232</td>
<td>4</td>
</tr>
<tr>
<td>PageRank</td>
<td>146</td>
<td>13</td>
</tr>
<tr>
<td>Mutual Neighbors</td>
<td>169</td>
<td>6</td>
</tr>
<tr>
<td>Connected Components</td>
<td>122</td>
<td>9</td>
</tr>
<tr>
<td>Triangles</td>
<td>181</td>
<td>6</td>
</tr>
<tr>
<td>Clustering Coefficients</td>
<td>218</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>1,068</td>
<td>50</td>
</tr>
</tbody>
</table>
Comparisons of Graph Frameworks

- In collaboration with Intel Parallel Research Lab
- Compared frameworks
  - SociaLite
  - Giraph
  - GraphLab
  - Combinatorial BLAS
- Native Implementation in C, assembly – optimal
Comparisons of Graph Frameworks

- Benchmark Algorithms
  - BFS (Breadth First Search)
  - PageRank
  - Collaborative Filtering
  - Triangle

- Evaluation on Intel cluster
  - Intel Xeon E5-2697, 24 cores 2.7GHz, 64GB memory, InfiniBand network
  - Input graph: up to 500M vertices, 16B edges
## Programmability

- **BFS (Breadth First Search)**

<table>
<thead>
<tr>
<th></th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socialite</td>
<td>4</td>
</tr>
<tr>
<td>Giraph</td>
<td>200</td>
</tr>
<tr>
<td>GraphLab</td>
<td>180</td>
</tr>
<tr>
<td>Combinatorial BLAS</td>
<td>450</td>
</tr>
<tr>
<td>Native</td>
<td>&gt; 1000</td>
</tr>
</tbody>
</table>
### BFS (Breadth First Search)

<table>
<thead>
<tr>
<th></th>
<th>Lines of Code</th>
<th>Development Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socialite</td>
<td>4</td>
<td>10min</td>
</tr>
<tr>
<td>Giraph</td>
<td>200</td>
<td>1~2 hours</td>
</tr>
<tr>
<td>GraphLab</td>
<td>180</td>
<td>1~2 hours</td>
</tr>
<tr>
<td>Combinatorial BLAS</td>
<td>450</td>
<td>a few hours</td>
</tr>
<tr>
<td>Native</td>
<td>&gt; 1000</td>
<td>&gt; A few months</td>
</tr>
</tbody>
</table>
Distributed Execution – Comparison

- **Breadth First Search**
  - Native
  - Combblas
  - Graphlab
  - Giraph
  - Time per iter. (sec.)

- **PageRank**
  - Time per iter. (sec.)

- **Collaborative Filtering**
  - Time per iter. (sec.)

- **Triangle**
  - Exec time (sec.)

### # of machines

- 1
- 4
- 16
- 64
Distributed Execution – Comparison

**Breadth First Search**

- **Native**
- **Combblas**
- **Graphlab**
- **Socialite**
- **Giraph**

- Execution time (sec.)
- Time per iter. (sec.)
- # of machines

**PageRank**

- Execution time (sec.)
- Time per iter. (sec.)

**Collaborative Filtering**

- Execution time (sec.)
- Time per iter. (sec.)

**Triangle**

- Execution time (sec.)
- Time per iter. (sec.)
Summary

- Large-scale graph analysis made easy with SociaLite
  - Succinct code (1/10th)
  - High-level abstraction w/ compiler optimizations
- Competitive performance to other graph frameworks
Thank you!

Questions?
Analysis of DBLP co-authorship graph

DBLP
– CS bibliography containing authors and papers
– Co-authorship graph
  1.2M nodes, 9.5M edges
Homework

- Example code
  - Shortest-paths
  - PageRank
- Part 1
- Part 2

→ Will be announced later today