Recursive Computation of Regions and Connectivity in Networks

**Mengmeng Liu**, Nicholas E. Taylor, Wenchao Zhou, Zachary G. Ives, Boon Thau Loo

Computer and Information Science
University of Pennsylvania
Motivation

• Distributed protocols and monitoring applications are very complex to implement!
  – Low-level details: synchronization, consistency, etc
  – Hard-wired to specific domains/apps
  – Programmed in procedural languages

• Motivating applications:
  – Router state: shortest paths, connectivity, etc
  – Sensor state: contiguous regions, etc

• Higher-level abstractions? Declarative queries?
  – Declarative Networking
  – TinyDB, Declarative Sensor Networks
Setting: Distributed Network State

- Network state is represented as distributed tables.
- Protocols are specified as declarative views over base relations.
- Expired link tuples become deletions in base relations.

### Base relation: link(src, dst)

### Derived relation: reachable(src, dst)

Reachability Query in Datalog:

R1: reachable(S,D) \(\rightarrow\) link(S,D).

R2: reachable(S,D) \(\rightarrow\) link(S,Z), reachable(Z,D).

Find me all destinations X can reach!
Distributed Computation

Base relation: link(src,dst)
Derived relation: reachable(src,dst)
Reachability Query in Datalog:
R1: reachable(S,D) :- link(S,D).
R2: reachable(S,D) :- link(S,Z), reachable(Z,D).

- Ship link tuple
- Join
- Ship reachable tuple
- Duplicate removal (under set semantics)
- Fixpoint
Other Interesting Recursive Queries

• Network reachability
  – Compute all reachable pairs of nodes

• Network shortest paths
  – Find the shortest path between pairs of nodes
  – Find the maximum bandwidth path between pairs of nodes
  – Find the most reliable path between pairs of nodes

• Sensing contiguous regions
  – Compute contiguous regions of triggered sensors
  – Find the largest region of triggered sensors
Roadmap

• Motivation and Applications

• Distributed Recursive Stream View Maintenance

• Novel techniques:
  – Annotate tuples with derivability information
  – Minimize propagation of tuples

• Experimental Evaluation

• Related Work

• Conclusion and Future Work
Distributed Recursive Stream View Maintenance

• How to incrementally compute the changes to the view when base tuples get updated?

• Difficult part: incremental deletion
  – A tuple in the view may be derived multiple times – when do we remove it?
  – For recursive case, a tuple may have infinite derivations...
Previous Incremental Deletion Solution: DRed [Gupta+ 93]

DRed approach: Over-delete and Re-derive

In many settings, DRed might aggressively delete all the tuples in the view and then recompute them all – too many network messages!
Our Focus

• Goal: minimize messages and bandwidth, while handing deletions

• Two sources of “redundant messages”
  – Over-deleted and re-derived tuples
  – Tuples that don’t contribute to the output (e.g., above MIN tuples)

• Our novel techniques:
  – Annotate tuples with derivability information
  – Buffer state necessary to handle deletions, and propagate as little of this state as possible
When Is a Tuple Derivable?

• **Idea**: annotating tuples with information on what tuples they were derived *from*

• Possible schemes:
  – Encode the relationship among tuples (graph)
  – Encode dependence on base tuples (condition)
Annotation with a Graph

• Relative Provenance [Green+07]
  – Graph to store derivations of one tuple directly from others
  – Traverse the graph to test derivability
Annotate with a Condition

• Annotate each tuple with a Boolean condition under which its existence depend on base tuples.
• We annotate base tuples with unique Boolean variables.
• Then we annotate derived tuples by the following rules:

<table>
<thead>
<tr>
<th>$t_1$</th>
<th>$p_1$</th>
<th>Joins:</th>
<th>$t_1 \bowtie t_2$</th>
<th>$p_1 \land p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
<td>$p_2$</td>
<td>Unions:</td>
<td>$t_1 \cup t_2$</td>
<td>$p_1 \lor p_2$</td>
</tr>
</tbody>
</table>

• Keep annotations compact by Absorption Law:

$$p_1 \land (p_1 \lor p_2) \equiv p_1 \lor (p_1 \land p_2) \equiv p_1$$

• Absorption Provenance
Incremental Deletion with Absorption Provenance

Annotate Tuples with Absorption Provenance.

Zero out the deleted provenance tokens in the Absorption Provenance.
How to Encode Absorption Provenance?

• Encode and maintain Boolean expressions while applying absorption

• Binary Decision Diagrams [Bryant 86]
  – Highly optimized libraries available: e.g. JavaBDD.

Boolean expression: \( \overline{x_1} \overline{x_2} x_3 + \overline{x_1} x_2 x_3 + x_1 x_2 \)
Roadmap

• Motivation and Applications

• Distributed Recursive Stream View Maintenance

• Novel techniques:
  – Annotate tuples with derivability information
  – Minimize propagation of tuples

• Experimental Evaluation

• Related Work

• Conclusion and Future Work
Challenge: We Now Propagate Tuples & Derivations, Not Just Tuples

Ship tuple (A,B) from Node A to Node B!
Lazy Propagation of Provenance

• We introduce a MinShip operator with lazy propagation of provenance
  – Idea: separate data from provenance
  – Only propagate the first derivation of a tuple, and buffer the subsequent derivations; trigger propagating buffered provenance only when necessary

Ship tuple (A,B) from Node A to Node B!

@B:

@A:

\[ \bigvee ( ) = P_2 \]
Minimizing Messages Involving Aggregate Computations

Shortest path query:
R1: \( \text{path}(S,D,C) : - \text{link}(S,D,C) \).
R2: \( \text{path}(S,D,C) : - \text{link}(S,Z,C_1), \text{path}(Z,D,C_2), C=C_1+C_2 \).
R3: \( \text{mincostpath}(S,D, \min<C>) : - \text{path}(S,D,C) \)

Buffer of path tuples:

Agg: \( \min<c> \)

Aggregate selection [Sudarshan+91]: prune away early tuples that don’t contribute to the results

Here: provenance-aware aggregate selection
Roadmap

• Motivation and Applications
• Distributed Recursive Stream View Maintenance
• Novel techniques:
  – Annotate tuples with derivability information
  – Minimize propagation of tuples
• Experimental Evaluation
• Related Work
• Conclusion and Future Work
Experimental Settings

• Set-up:
  – Java-based distributed query processor
  – 24-node cluster, Intel Xeon 2.4GHz PCs with 4GB RAM running Linux

• Workload: simulated Internet topologies:
  – GT-ITM [GA Tech]: standard simulator from network community
  – Reachability query, 100 nodes, 200 bi-directional edges

• Methods in comparison:
  – Lazy propagation (MinShip) / Eager propagation (Ship)
Deletion Workloads

communication traffic (MB)

<table>
<thead>
<tr>
<th>Deletion Ratio</th>
<th>DRed</th>
<th>Relative Lazy</th>
<th>Absorption Lazy</th>
<th>Absorption Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

per-tuple size (B)

<table>
<thead>
<tr>
<th>Deletion Ratio</th>
<th>DRed</th>
<th>Relative Lazy</th>
<th>Absorption Lazy</th>
<th>Absorption Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary of Experimental Results

• Absorption provenance
  – Huge savings of bandwidth (20x) for deletions
  – Reasonable per-tuple provenance storage overhead

• Similar results:
  – Region queries in sensor networks

• Shortest path query and largest region query:
  – Great advantages of provenance-aware aggregate selection (10x bandwidth efficiency on sparse graphs and 100x on dense graphs)
  – Multi-aggregate selection 2x better than single one
Related Work

• Declarative Networks and Declarative Sensor Networks
  – P2: declarative networking [Loo+06]
  – TinyDB [Madden+03], DSN [Chu+06]

• Incremental View Maintenance
  – DRed [Gupta+93]

• Aggregate Selection [Sudarshan+91]

• Data Provenance
  – Relative Provenance [Green+07a]
  – Provenance Semirings [Green+07b]
Conclusion and Future Work

• Distributed recursive stream view maintenance
  – Annotate tuples with derivability information
    • Absorption provenance
  – Minimizing propagation of tuples
    • Minship and lazy propagation
    • Provenance-aware aggregate selection

• Future work
  – Optimizing with absorption provenance
  – Cost-based query optimization