Time-evolving Graph Processing on Commodity Clusters

SIAM CSE 17
2 March 2017

Anand Iyer  Qifan Pu  Joseph Gonzalez  Ion Stoica
Graphs are everywhere...
Graphs are everywhere...

Gnutella network subgraph
Graphs are everywhere...
Graphs are everywhere...

Metabolic network of a single cell organism

Tuberculosis
Plenty of interest in processing them

- Graph DBMS 25% of all enterprises by end of 2017\(^1\)
- Many open-source and research prototypes on distributed graph processing frameworks: Giraph, Pregel, GraphLab, GraphX, ...

\(^1\)Forrester Research
Real-world Graphs are Dynamic

Earthquake Occurrence Density
Real-world Graphs are Dynamic
Real-world Graphs are Dynamic
Many interesting business and research insights possible by processing such dynamic graphs...

... little or no work in supporting such workloads in existing big-data graph-processing frameworks
Challenge #1: Storage

Redundant storage of graph entities over time
Challenge #2: Computation

Wasted computation across snapshots
Challenge #3: Communication

Duplicate messages sent over the network
How do we process time-evolving, dynamically changing graphs efficiently?
How do we process time-evolving, dynamically changing graphs efficiently?
Sharing Storage

Storing deltas result in the most optimal storage, but creating snapshot from deltas can be expensive!
A Better Storage Solution

Use a persistent datastructure

Store snapshots in Persistent Adaptive Radix Trees (PART)
Graph Snapshot Index

Snapshot ID Management

Shares structure between snapshots, and enables efficient operations
How do we process time-evolving, dynamically changing graphs efficiently?
Graph Parallel Abstraction - GAS

**Gather**: Accumulate information from neighborhood

**Apply**: Apply the accumulated value

**Scatter**: Update adjacent edges & vertices with updated value
for (snapshot in snapshots) {
    for (stage in graph-parallel-computation) {...}
}
Reducing Redundant Messages

for (step in graph-parallel-computation) {
    for (snapshot in snapshots) {...}
}

Can potentially avoid large number of redundant messages
How do we process time-evolving, dynamically changing graphs efficiently?
Updating Results

- If result from a previous snapshot is available, how can we reuse them?
- Three approaches in the past:
  - Restart the algorithm
    - Redundant computations
  - Memoization (GraphInc$^1$)
    - Too much state
  - Operator-wise state (Naiad$^{2,3}$)
    - Too much overhead
    - Fault tolerance

$^1$Facilitating real-time graph mining, CloudDB ’12
$^2$Naiad: A timely dataflow system, SOSP ’13
$^3$Differential dataflow, CIDR ’13
Key Idea

- Leverage how GAS model executes computation
- Each iteration in GAS modifies the graph by a little
  - Can be seen as another time-evolving graph!
- Upon change to a graph:
  - Mark parts of the graph that changed
  - Expand the marked parts to involve regions for recomputation in every iteration
  - Borrow results from parts not changed
Incremental Computation

Larger graphs and more iterations can yield significant improvements
Implementation & Evaluation

- Implemented on Spark 2.0
  - Extended dataframes with versioning information and iterate operator
  - Extended GraphX API to allow computation on multiple snapshots
- Preliminary evaluation on two real-world graphs
  - **Twitter**: 41,652,230 vertices, 1,468,365,182 edges
  - **uk-2007**: 105,896,555 vertices, 3,738,733,648 edges
Benefits of Storage Sharing

- Datastructure overheads
- Significant improvements with more snapshots

Storage Reduction vs. Number of Snapshots
Benefits of sharing communication

![Graph showing benefits of sharing communication](image)
Benefits of Incremental Computing

Only 5% of the graph modified in every snapshot

50x reduction by processing only the modified part
Summary & Future Work

• Processing time-evolving graph efficiently can be useful
• Sharing storage, computation and communication key to efficient time-evolving graph analysis
• Code release
• Incremental pattern matching
• Approximate graph analytics
• Geo-distributed graph analytics

api@cs.berkeley.edu
www.cs.berkeley.edu/~api