Introduction

- **Graphs** are widely considered to be natural means of representation for many networks.
- Real-world networks are often evolving with links being added or removed and properties updated over time.

Examples: Social, Citation/Collaboration, Sensor, Financial & Transit Network, Human Connectomes, Internet-of-Things ...

- **Incremental computation** is used to process such dynamic graphs. It achieves efficiency by minimizing redundant computation and communication compared to complete, from-scratch computation.

**This Work**: General-purpose distributed programming model to support incremental processing over dynamic graphs while leveraging existing vertex-centric semantics.

Incremental Graph Processing

- Better state re-use
- Minimized redundant computation and communication
- Faster convergence time

Experimental Evaluation

- **Graph**: Twitter (TW)
- Vertices: 317M
- Edges: 31B

Wave Insert (Delete 50:50)

Execution time (in seconds) for Recompute, TAG, Kickstarter-Lite, Wave for 100K mutations

Wave is 6-23x faster than recomputation, 4-14x faster than TAG, and 2-6x faster than KS-Lite

Summary

- Incremental Graph Processing
- Better state re-use
- Minimized redundant computation and communication
- Faster convergence time

Incremental Graph Computation

- **Transparency**: We do not require the vertex program to be made incremental, but instead incrementalize its boundary.
  - Makes the approach applicable to all existing vertex programs
  - Requires minimal additional effort from the programmer to devise an incremental algorithm

- Framework identifies vertices directly and transitively affected by graph mutations.
  - Only parts of the graph affected by input changes are re-computed
  - Graph structure used to actively deduce value dependencies

Connected Components

- Vertex updates over supersteps when finding Connected Components. Changes shown in red.