Pregel: A System for Large-Scale Graph Processing

Enes Goktas
Outline

• Introduction
• Model
• Application
• Experiment results
• Conclusion
Introduction 1/2

• Large-scale graphs to process

• Typical ways
  – Creating custom distributed infrastructure
  – Using existing distributed computing platform
  – Using single-computer graph algorithm library
  – Using existing parallel graph system
Introduction 2/2

• All have issues processing large-scale graphs

• Pregel
  – Address large-scale graphs
  – Scalable
  – Fault tolerant
  – Flexible API
Model

- Supersteps
- Vertex-centric approach
- Master and workers
- Termination
- Fault tolerance
Model: Supersteps

• A sequence of iterations

• A master instructs workers

• Vertices execute same function in parallel
Model: Vertex-centric

- Initially user defined value
- Modify value: itself / edge
- Receive / send message
- Mutate graph’s topology
Model: Master & Workers

• Master
  – coordinates workers

• Workers
  – Iterate its vertices
  – Calls the function (Compute())
Model: Termination 1/2

- Vertex state machine:
Model: Termination 2/2

Superstep 0

Superstep 1

Superstep 2

Superstep 3
Model: Fault tolerance

- Master instructs workers
- Checkpointing
- Ping
  - To mark or not
- At fail, reload from most recent checkpoint
Application

• Single-source shortest path

• Vertex recv distances to source from neighbors

• Compute the minimal distance

• If necessary
  – Update own value
  – Send updated value to neighbors

• Vote to halt
Experiment: set up

• App: Single-source shortest path
• 300 multicore commodity PCs
• Edge weight: 1

• Scalability: Binary tree

• Performance: More realistic graph
Experiment: Scalability 1/2

- Vary number of workers
- Binary tree
  - Billion vertices
  - Billion-1 edges
- 50 -> 800 w
- 174 -> 17.3 s
- -> 10x

![Graph showing runtime vs. number of worker tasks]
Experiment: Scalability 2/2

- Vary graph size
- Fixed 800 workers
Experiment: Performance

- Binary tree not representative
- More realistic graph
  - Mean out-degree 127.1
  - 800 workers

![Graph showing runtime vs number of vertices]
Conclusion

• Little programming effort

• Billions of vertices

• Satisfactory performance

• Easy if you “think as a vertex”