Good morning!

CS 744: POWERGRAPH

Shivaram Venkataraman
Fall 2021
ADMINISTRIVIA

- Midterm grading in progress
- Course Project
  - Checkpoint update = end of Nov
  - Office hours / setup meetings
Scalable Storage Systems
Computational Engines
Resource Management
Datacenter Architecture
GRAPH DATA

Datasets
- Twitter / Social network
  - graph of follows/friends
- Web graph
  - links between pages
- Maps
  - locations connected by streets
- Facts about entities [Wikipedia]
  → knowledge

Application
- Recommendation
  - "You might know"
- Ranking / Scoring
  - PageRank
- Directions / Traffic analysis
GRAPH ANALYTICS

Perform computations on graph-structured data

Examples
  PageRank
  Shortest path
  Connected components
  ...

Online graph serving
  - low latency traversals
  - graph updates

Analytics
  - batch job
  - large graph and
    you want to analyze
    it
Message combiner(Message m1, Message m2):
    return Message(m1.value() + m2.value());

void PregelPageRank(Message msg):
    float total = msg.value();
    vertex.val = 0.15 + 0.85*total;
    foreach(nbr in out_neighbors):
        SendMsg(nbr, vertex.val/num_out_nbrs);

"think like a vertex"

vertex Program that operates on messages over Edges
NATURAL GRAPHS

- Skew in the "in-degree"
- So very few users have lots of followers
- Some vertices have lots of messages come in
- Work Imbalance → "Compute"
- Storage / Network →

Stragglers / how utilization / Increased time for 1 iteration

(a) Twitter In-Degree
POWERGRAPH

Programming Model:
Gather-Apply-Scatter

Sync / Async execution

Better Graph Partitioning
with vertex cuts

→ vertex based programming model
GATHER-APPLY-SCATTER

Gather: Accumulate info from nbrs

Apply: Accumulated value to vertex

Scatter: Update adjacent edges

// gather_nbrs: IN_NBRS
gather(Du, D(u,v), Dv):
    return Dv.rank / #outNbrs(v)

sum(a, b): return a + b = combiner

apply(Du, acc):
    rnew = 0.15 + 0.85 * acc
    Du.delta = (rnew - Du.rank) / #outNbrs(u)
    Du.rank = rnew

// scatter_nbrs: OUT_NBRS
scatter(Du, D(u,v), Dv):
    if(|Du.delta| > ε) Activate(v)
    return delta
EXECUTION MODEL, CACHING

1. Activate all vertices
   not all active in every iteration

Active Queue

... V2 V1 ...

Delta caching
   Cache accumulator value for vertex
   Optionally scatter returns a delta
   Accumulate deltas
   \[ \text{Saves a lot of gather calls} \]
**SYNC VS ASYNC**

**Sync Execution**
- Gather for all active vertices,
- followed by Apply, Scatter
- Barrier after each minor-step

**Async Execution**
- Execute active vertices,
- as cores become available
- No Barriers! Optionally serializable

\[ G(v_1) \rightarrow A(v_1) \rightarrow \text{update } v_1 \text{ state} \]

\[ G(v_2) \leftarrow \text{reads updated state} \]

- Update vertex & edge state “eagerly”
- Some algorithms accelerates
- No guarantees on convergence
DISTRIBUTED EXECUTION

Symmetric system, no coordinator

Load graph into each machine

Communicate across machines to spread updates, read state
GRAPH PARTITIONING

(a) Edge-Cut
- Place a vertex on a machine
- Minimize number of edges that cross machines
- Can lead to imbalance

(b) Vertex-Cut
- Place an edge on a machine
- Replicas of vertex state when edges are on diff machines
- One primary replica!

ghost vertices

place a vertex on a machine

Minimize number of edges that cross machines

Can lead to imbalance
RANDOM, GREEDY OBLIVIOUS

Three distributed approaches:

Random Placement

- Stream through edges, pick a random machine

Coordinated Greedy Placement

- Check which machine already has this vertex and place edge there

Oblivious Greedy Placement

- Only know local set of vertices, not global
OTHER FEATURES

Async Serializable engine
  Preventing adjacent vertex from running simultaneously
  Acquire locks for all adjacent vertices

Fault Tolerance
  Checkpoint at the end of super-step for sync
SUMMARY

Gather-Apply-Scatter programming model
Vertex cuts to handle power-law graphs
Balance computation, minimize communication
DISCUSSION

https://forms.gle/Xs3ibsUCdjynBv7u8
Consider the PageRank implementation in Spark vs synchronous PageRank in PowerGraph. What are some reasons why PowerGraph might be faster?

- Delta caching
  → Communication, computation might be lower
  → Join between edge list and PageRank

- Vertex Cuts
  → Edge cuts in Spark. Imbalance / more communication
  Join step

- Fault tolerance
  → Partial recovery can be faster?
Coord has best iteration time

Comm keeps going up!

wins are not big after 32 machines

(a) Twitter PageRank Runtime

(b) Twitter PageRank Comms
NEXT STEPS

Next class: GraphX

/ COST

which sections of which papers