Welcome back!

CS 744: POWERGRAPH

Shivaram Venkataraman
Fall 2022
- Midterm grading in progress → end of this week
- Course Project: GCP credits
Scalable Storage Systems

Datacenter Architecture

Resource Management

Computational Engines

Applications

Machine Learning  SQL  Streaming  Graph

Applications

Machine Learning  SQL  Streaming  Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture
GRAPH DATA

Datasets
- Web graph: hyperlinks
- Social network graph
- Maps: Locations → vertices, Roads → edges
- Knowledge graph
  - Obama was president 2008-2014
- Chemicals / Molecules

Application
- Search (important or relevant pages)
- Recommendation
- Classify chemical structures
GRAPH ANALYTICS

Perform computations on graph-structured data

Examples
- PageRank
- Shortest path
- Connected components
- ...

Classic
- PageRank
- Connected Components

Apply ML on graph structured data

"OLTP"

serving graph data

traverse graph

return results

latency sensitive

Analytics
PREGEL: PROGRAMMING MODEL

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);

1. Express graph algorithms,
2. Recv inputs from nbrs → Comp on the vertex → Send msg to nbrs
NATURAL GRAPHS

1. Exp. distribution of nbrhood sizes
   - few vertices very large degree
   - large number with very small degree

2. Lack of symmetry
   - Imbalance in work done per vertex
   - Imbalance lead to low utilization
     \[\Rightarrow\] stragglers

(a) Twitter In-Degree
\[\alpha = 1.7\]
POWERGRAPH

Programming Model:
Gather-Apply-Scatter

Sync / Async execution

Better Graph Partitioning
with vertex cuts

→ builds on think like a vertex

→ single machine

→ Distributed
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

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

At beginning
- Activate all vertices

Gather
- Gather \( V_0 \)
  - List all neighbors \( (V_0) \)
  - Get their state
  - Accumulate

Apply
- Apply \( V_0 \)
  - Read acc for \( V_0 \)
  - Comp.
  - Read state for \( V_0 \)
  - Write back updated

Scatter
- Activate neighbors in next iteration

Active Queue
- \( v_n \), \( v_i \), \( v_o \)

Accumulators
- Id
- State
- 0
- 0.15
CACHING

Active Queue

Accumulators

Persist across

Vertex State

Accumulators

Delta caching

Cache accumulator value for vertex

Optionally scatter returns a delta

Accumulate deltas

Avoid running gather for that vertex

Reuse acc computed in previous iteration

If few nbrs have changed
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

All the reads happen before updates.
DISTRIBUTED EXECUTION

Symmetric system, no coordinator

Partition graph across machines
Communicate to spread updates, read state
GRAPH PARTITIONING

(a) Edge-Cut
- Assign a vertex to a machine
- Implies edges are cut across machines. Minimize this
- Induce imbalance for natural graphs

(b) Vertex-Cut
- Assign edge to particular machine
- When vertex is on many machines, one primary
RANDOM, GREEDY OBLIVIOUS

Three distributed approaches:

Random Placement

- Given an edge, choose random machine

Coordinated Greedy Placement

- If either vertex is already placed, favor those machines

Oblivious Greedy Placement

- Avoid coordination, only track vertices present locally
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/K7xk2KybTXf3XX3b6
Consider the PageRank implementation in Spark vs synchronous PageRank in PowerGraph. What are some reasons why PowerGraph might be faster?

1. You can activate a subset of vertices across iterations ⇒ work goes down as iters progress

2. Partition in Spark ⇒ randomly partitions

   smarter partitioning here ? ⇒ lower communication

3. Delta caching ⇒ computation for active vertices can be lowered.
Partition time?

Comm goes up with num machine

Sub-linear with more machines

Diminishing returns get close to same

(a) Twitter PageRank Runtime

(b) Twitter PageRank Comms
NEXT STEPS

Next class: Marius