CS 744: POWERGRAPH

Shivaram Venkataraman
Fall 2019
ADMINISTRIVIA

- Midterm grades (end of) this week
- Course Projects sign up for meetings
- Google Cloud credits
Scalable Storage Systems

Datacenter Architecture

Applications

Machine Learning
SQL
Streaming
Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture

Applications

Machine Learning
SQL
Streaming
Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture

Applications

Machine Learning
SQL
Streaming
Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture

Applications

Machine Learning
SQL
Streaming
Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture
Graph Data

Datasets
- Friendship among users
- Knowledge base graph
  - Entity: Berlin, capital of Germany
- Microservice / graph of systems
- Map to locations & streets joining
- Internet -> hosts & links

Application
- Recommend new friends
- Question answering
- Debugging / root cause analysis
- Routing
  - Internet routing of Internet packets
Perform computations on graph-structured data

Examples
- PageRank
- Shortest path
- Connected components

...
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);

<table>
<thead>
<tr>
<th>Node</th>
<th>Neighboring</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NATURAL GRAPHS

- Degree distribution is very skewed
- Imbalance
  - Computation
  - Messages/communication
  - State vs num of neighbors

(a) Twitter In-Degree

\[ \alpha = 1.7 \]
POWERGRAPH

Programming Model:
Gather-Apply-Scatter

Better Graph Partitioning
with vertex cuts

Distributed execution (Sync, Async)
GATHER-APPLY-SCATTER

Gather: Accumulate info from nbrs

Apply: Accumulated value to vertex

Scatter: Update adjacent edges, vertices

Edge State, Vertex State

// 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, CACHING

Active Queue

Delta caching
  Cache accumulator value for vertex
  Optionally scatter returns a delta
  Accumulate deltas

Consistency
  - Visible to gather?

Optimization to reduce number of gather

Accumulate deltas
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
DISTRIBUTED EXECUTION

Symmetric system, no coordinator

Load graph into each machine

Communicate across machines to spread updates, read state
GRAPH PARTITIONING

(a) Edge-Cut

- No split in computation → vertex is in 1 machine
- When split edge → track vertex state across split
- Vertex has lots of edges, imbalance

(b) Vertex-Cut

- Edge is in 1 machine
- Vertices can be split across machines
Three distributed approaches:

Random Placement

- Data loading
- Stream placement edges on machines
- Fast data loading

Coordinated Greedy Placement

- Place edges in same machine that already has other edges with this vertex

Oblivious Greedy Placement

- Avoid this synchronization while data loading

<table>
<thead>
<tr>
<th>V</th>
<th>mUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>1,3,7</td>
</tr>
<tr>
<td>V2</td>
<td>3,4</td>
</tr>
</tbody>
</table>

\[ V \quad \text{mUS} \quad 5 \quad 1,3,7 \quad 3,4 \]
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
  For Async?
DISCUSSION

https://forms.gle/t2TJ4sEFDNZ8aDBo7
Consider the PageRank implementation in Spark vs synchronous PageRank in PowerGraph. What are some reasons why PowerGraph might be faster?

- Both are computing sync updates
- Fine grained parallelism
- Less communication from vertex cuts
- Not all nodes are activated
- State is mutable

Partioning function in Spark?
Synchronous(Random) = Suboptimal
Synchronous(Oblivious) = More
Synchronous(Coord.) = Random

(a) Twitter PageRank Runtime
(b) Twitter PageRank Comms
(c) Twitter PageRank Delta Cache

Delta caching improves perf

Activation based on delta

Activation, Compl. gather

Vertex
What could be one shortcoming of PowerGraph compared to prior systems like MapReduce or Spark?

- Specialized system
  Cross vertex analytics that is harder?
- Fault tolerance → checkpoint / restart?
- Stragglers?
Next class: GraphX
Sign up for project check-ins!