

CS 744: POWERGRAPH

Shivaram Venkataraman

Fall 2022

ADMINISTRIVIA

- Midterm grading in progress
- Course Project: GCP credits

Applications

Machine Learning

SQL

Streaming

Graph

Computational Engines

Scalable Storage Systems

Resource Management



Datacenter Architecture



GRAPH DATA

Datasets

Application

GRAPH ANALYTICS

Perform computations on graph-structured data

Examples

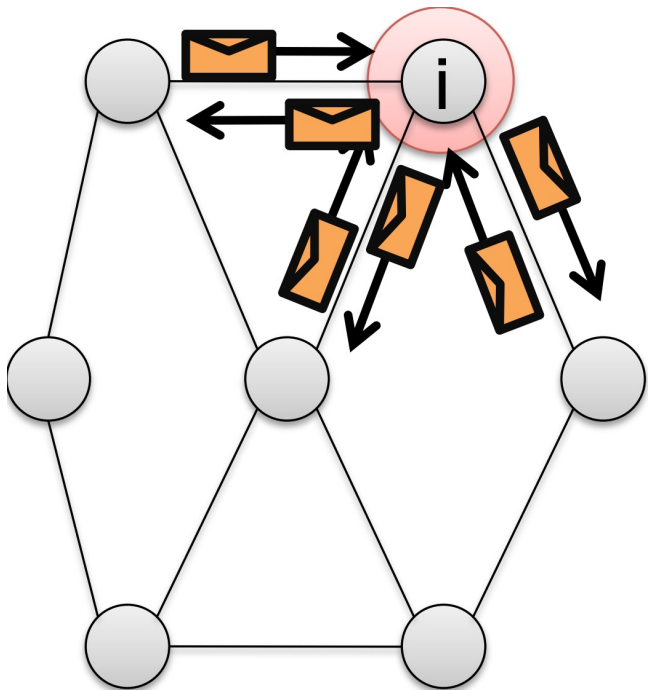
- PageRank

- Shortest path

- Connected components

- ...

PREGEL: PROGRAMMING MODEL



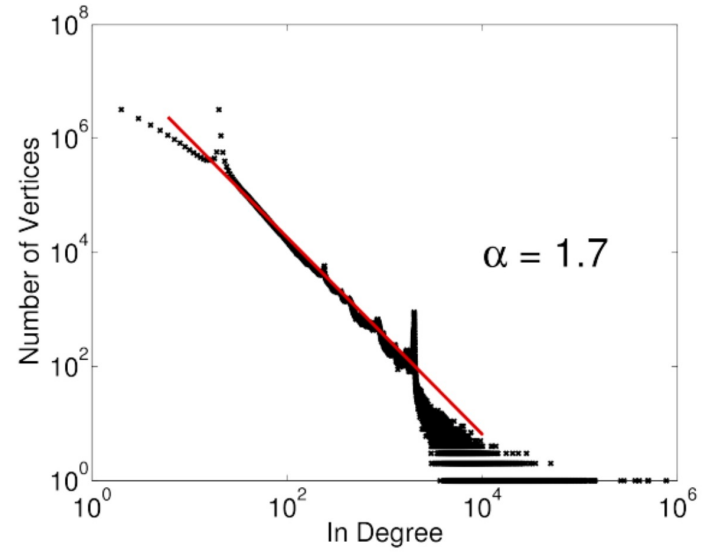
```
Message combiner(Message m1, Message m2):  
    return Message(m1.value() + m2.value());
```

```
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);
```

NATURAL GRAPHS



(a) Twitter In-Degree

POWERGRAPH

Programming Model:
Gather-Apply-Scatter

Sync / Async execution

Better Graph Partitioning
with vertex cuts

GATHER-APPLY-SCATTER

Gather: Accumulate info from nbrs

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

Apply: Accumulated value to vertex

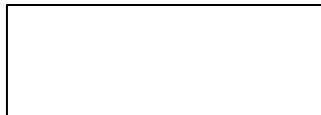
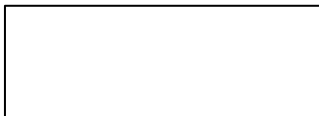
```
sum(a, b): return a+b
```

Scatter: Update adjacent edges

```
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

Accumulators



Vertex State

Active Queue



Gather

Apply

Scatter



CACHING

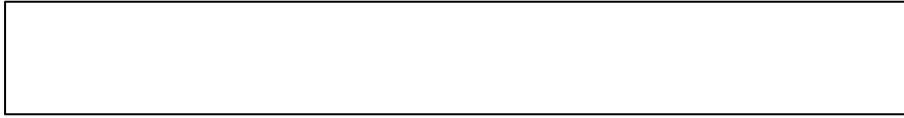
Accumulators



Vertex State



Active Queue



Delta caching

Cache accumulator value for vertex

Optionally scatter returns a delta

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

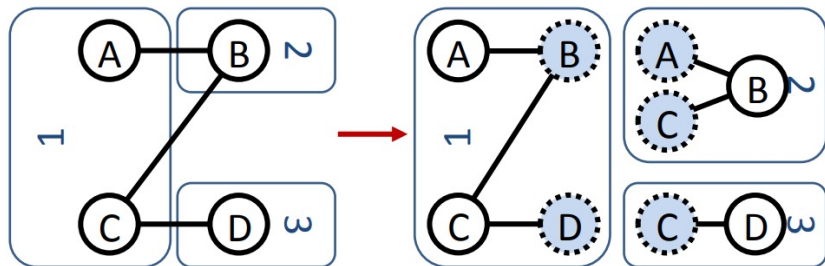
DISTRIBUTED EXECUTION

Symmetric system, no coordinator

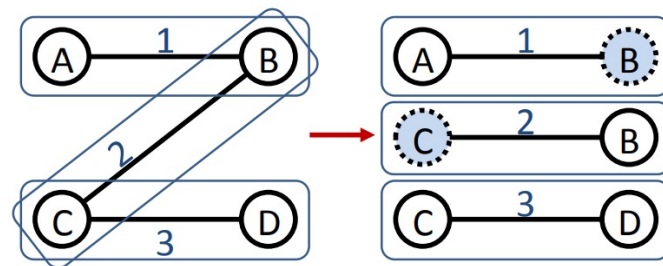
Partition graph across machines

Communicate to spread updates, read state

GRAPH PARTITIONING



(a) Edge-Cut



(b) Vertex-Cut

RANDOM, GREEDY OBLIVIOUS

Three distributed approaches:

Random Placement

Coordinated Greedy Placement

Oblivious Greedy Placement

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

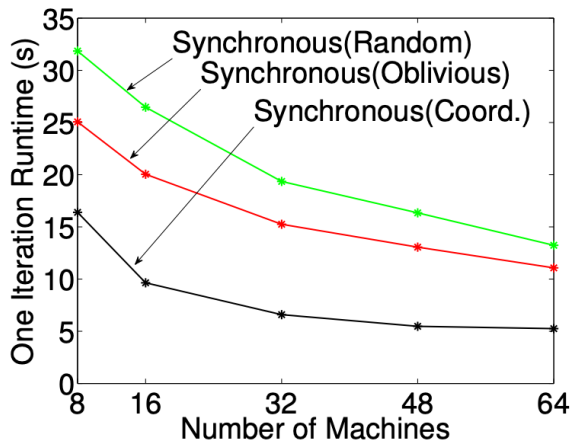
Vertex cuts to handle power-law graphs

Balance computation, minimize communication

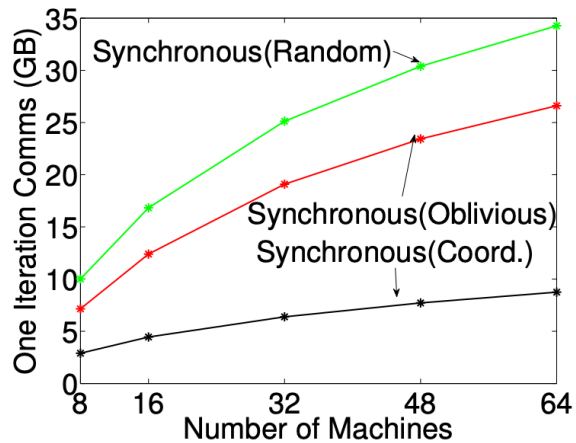
DISCUSSION

<https://forms.gle/K7xk2KybTXf3XX3b6>

Consider the PageRank implementation in Spark vs synchronous PageRank in PowerGraph. What are some reasons why PowerGraph might be faster?



(a) Twitter PageRank Runtime



(b) Twitter PageRank Comms

NEXT STEPS

Next class: Marius