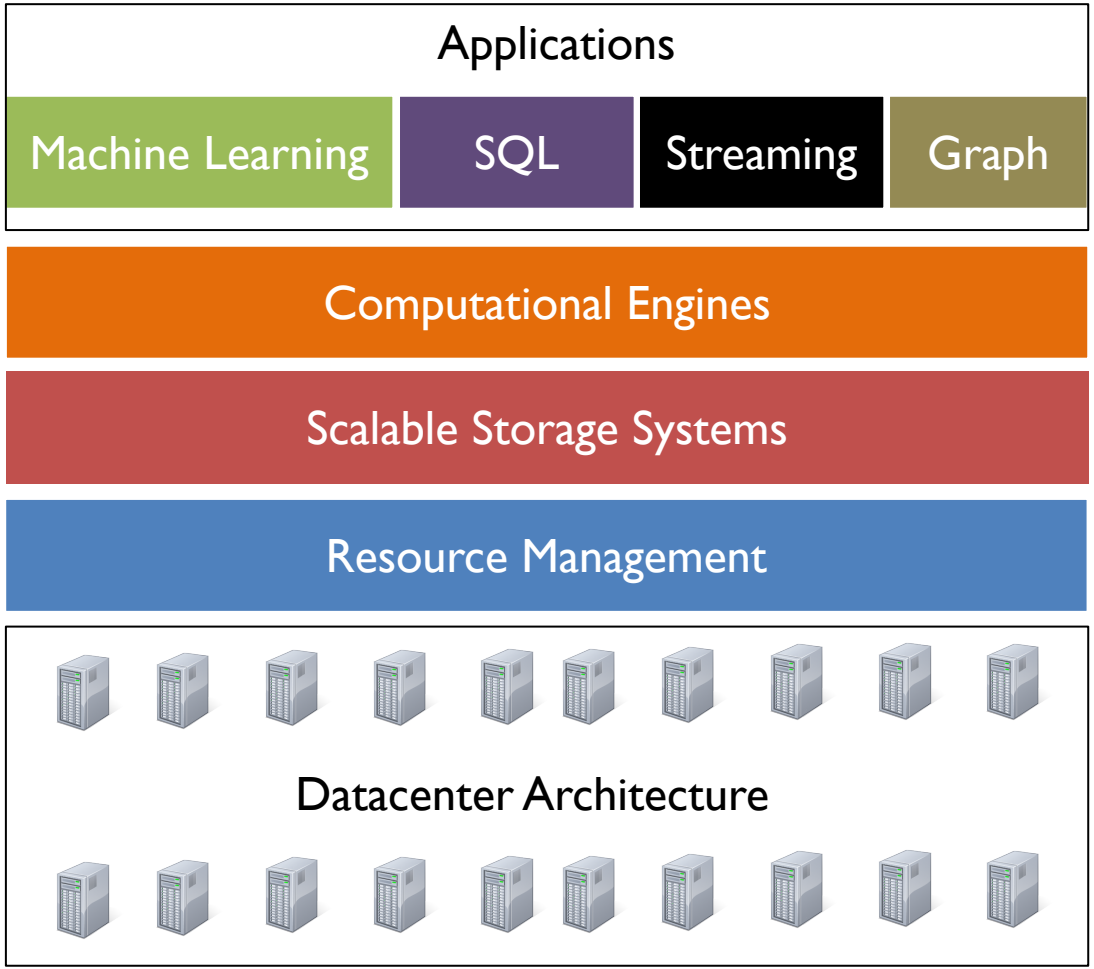
CS 744: POWERGRAPH

Shivaram Venkataraman

Fall 2020

ADMINISTRIVIA

- Midterm update
- Course Project reminders


Applications
Machine Learning
SQL
Streaming
Graph
Computational Engines
Scalable Storage Systems
Resource Management
Datacenter Architecture

GRAPH DATA

Datasets

Application

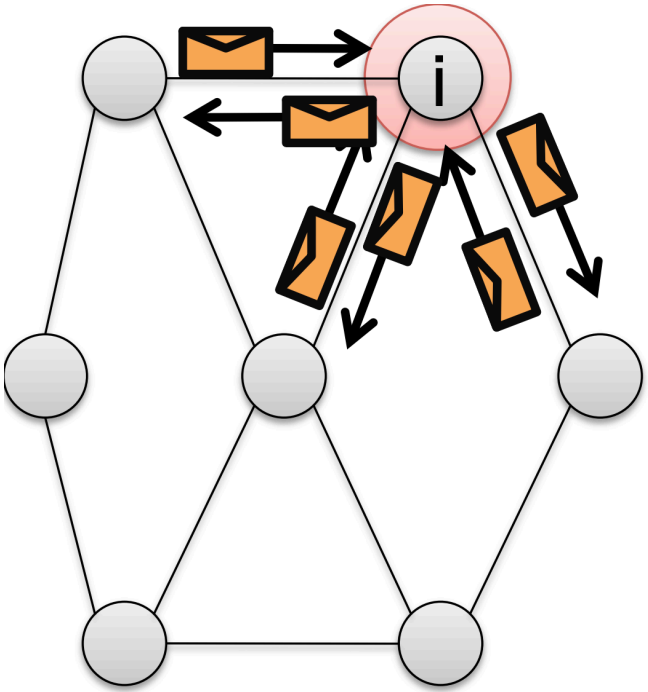
GRAPH ANALYTICS

Perform computations on graph-structured data

Examples

- PageRank
- Shortest path
- Connected components
- …

PREGEL: PROGRAMMING MODEL



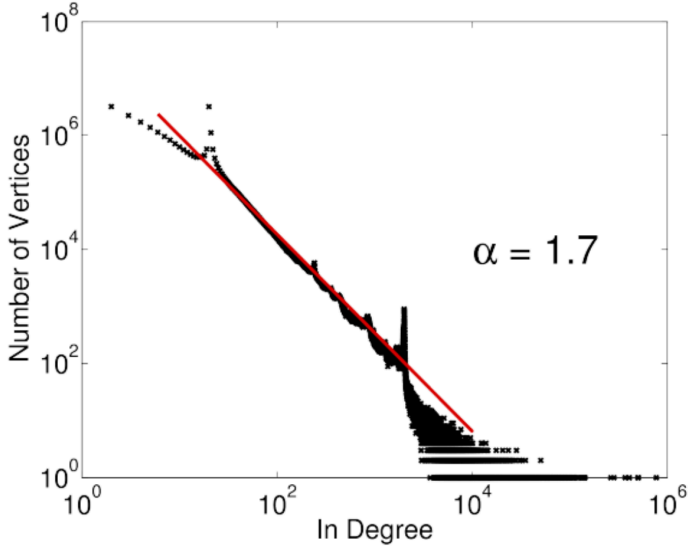

```
Message combiner(Message m1, Message m2):
    return Message(m1.value() + m2.value());

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

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

```
// 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

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

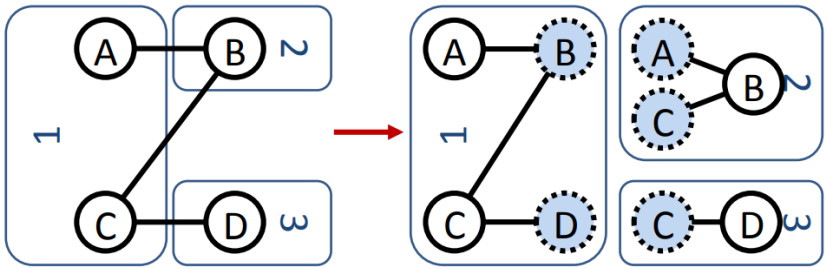
DISTRIBUTED EXECUTION

Symmetric system, no coordinator

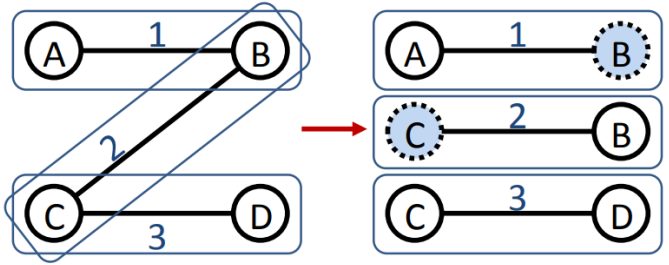
Load graph into each machine

Communicate across machines to spread updates, read state

GRAPH PARTITIONING



(a) Edge-Cut



(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

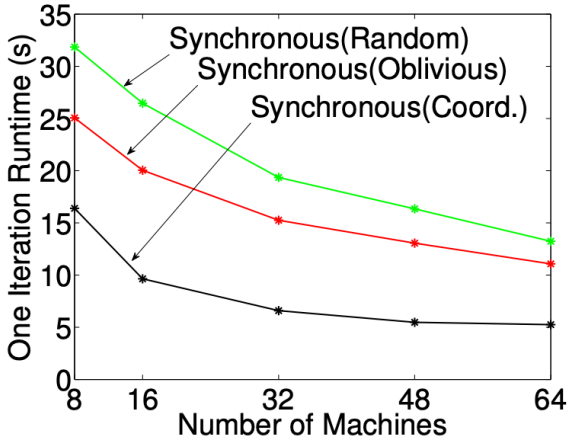
Vertex cuts to handle power-law graphs

Balance computation, minimize communication

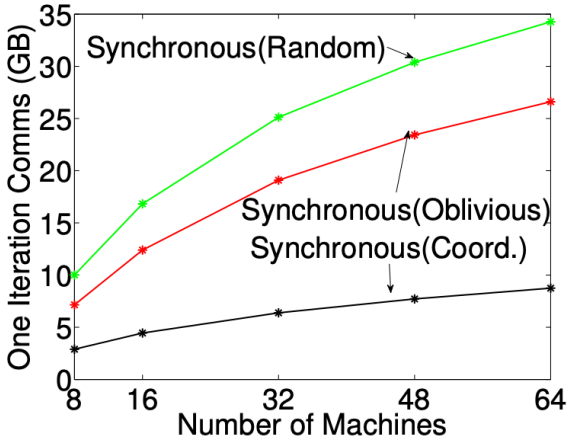
DISCUSSION

https://forms.gle/rKB5hcJgT4NQsFgq8

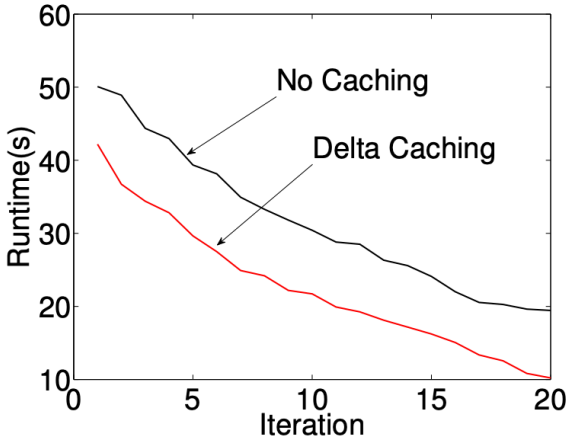
Consider the PageRank implementation in Spark vs synchronous PageRank in PowerGraph. What are some reasons why PowerGraph might be faster?



(a) Twitter PageRank Runtime



(b) Twitter PageRank Comms



(c) Twitter PageRank Delta Cache

NEXT STEPS

Next class: GraphX