Hello

CS 744: POWERGRAPH

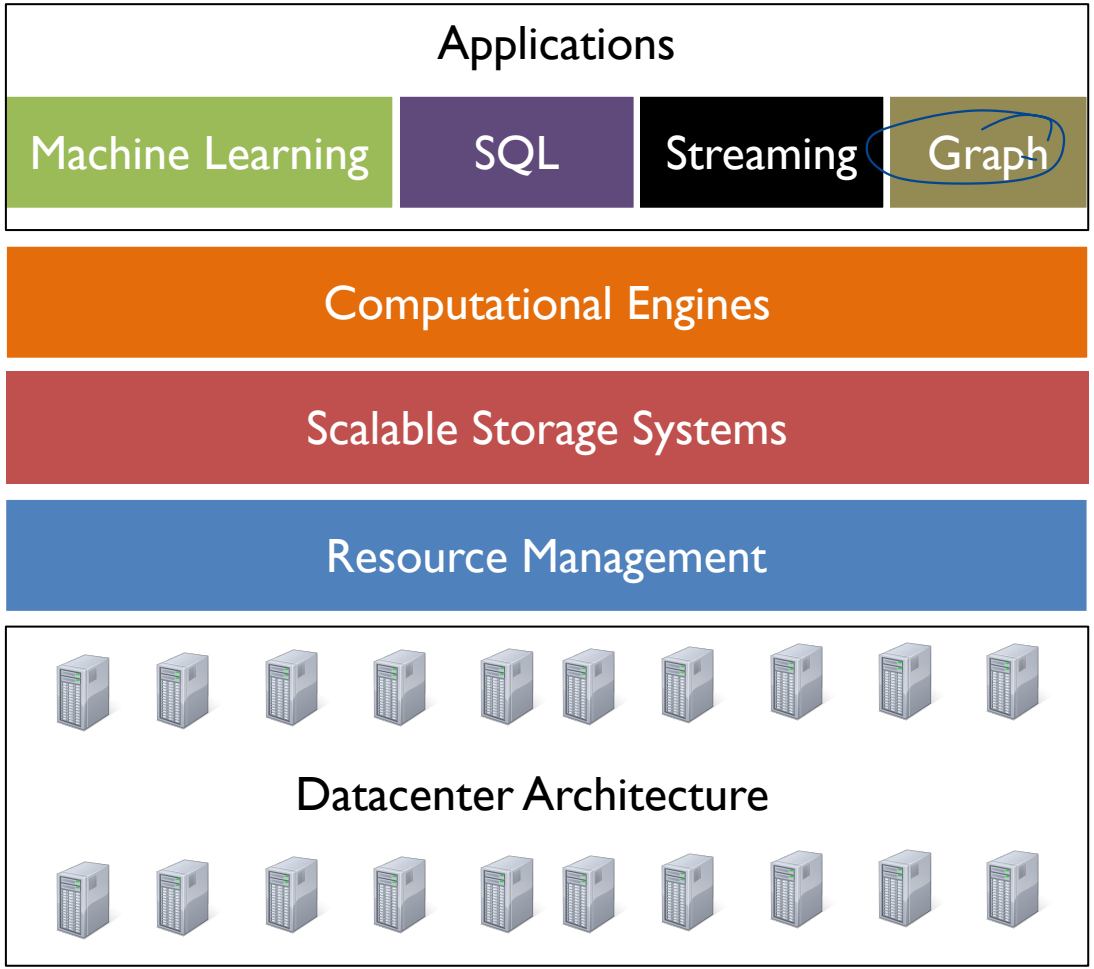
Shivaram Venkataraman

Spring 2024

ADMINISTRIVIA

- Midterm grading in progress → end of this week
- Cloudlab, GCP details
  - Reservations → 2 weeks
  - Redeeming credits → TA post on Piazza

- Check-ins
  - ~ April 16th ?
  - → Template / guidance on Piazza



graph structured data

- Page Rank

- ML algorithms on graphs

GRAPH DATA

Search results in Siri etc.

Datasets

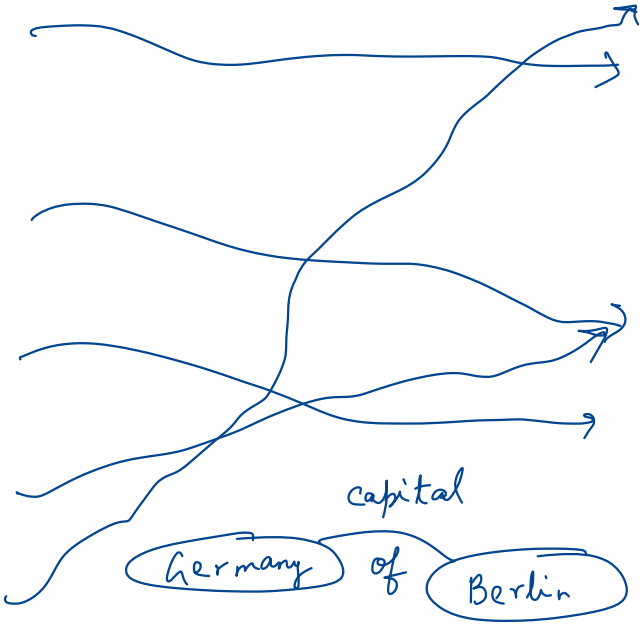
Social networks

Road networks

Protein structure

Internet topology

Knowledge graph



Application

Predict link

Recommend friends / products

Shortest path

Synthesize new proteins

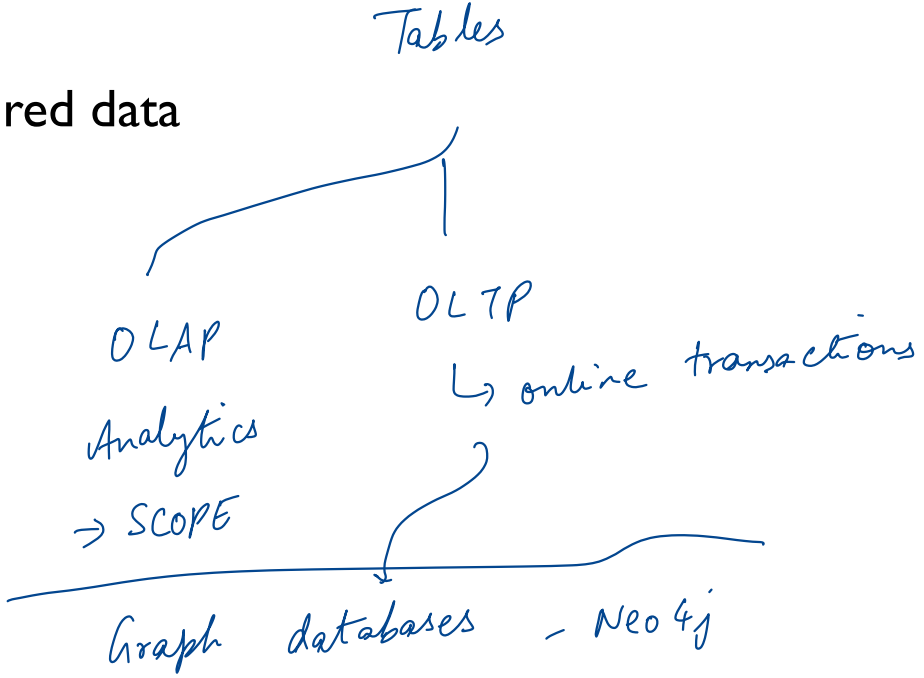
GRAPH ANALYTICS

Perform computations on graph-structured data

Examples

- PageRank
- Shortest path
- Connected components
- …

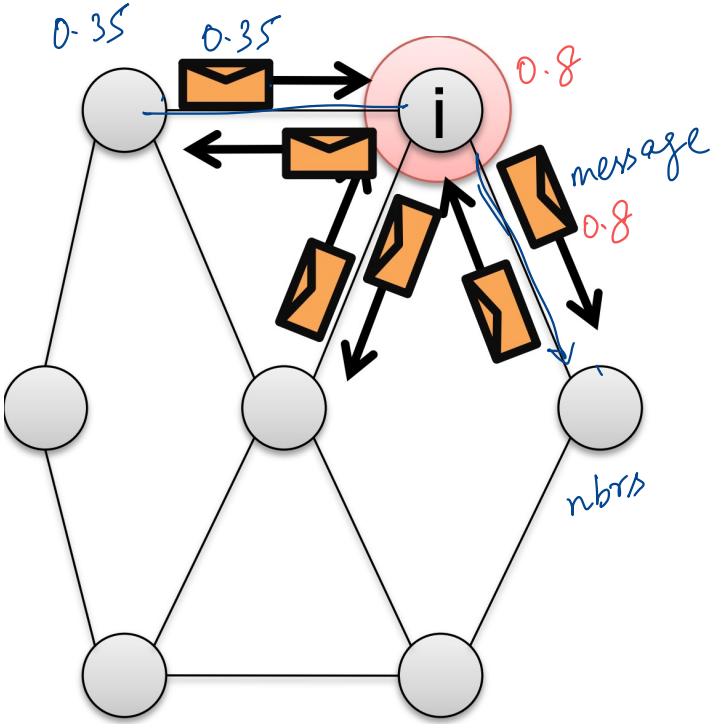
→ Analytics on graphs



~2008 - 2009

PREGEL: PROGRAMMING MODEL

"MapReduce for graphs"




```
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);
```

incoming message

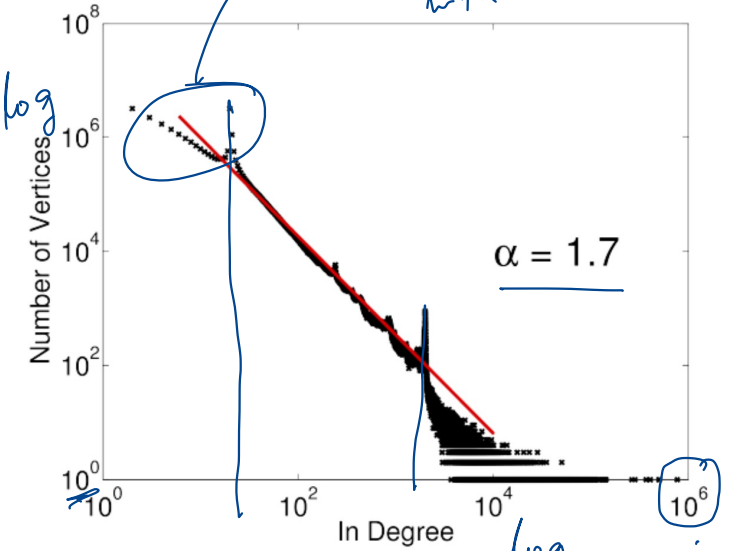
"think - like - a - vertex"

update vertex state

send to nbrs

NATURAL GRAPHS

- Skew
  - Some nodes have a lot of neighbors (upto 1M nbrs)
    - ↳ lot of computation / messages for this high degree vertex
  - Imbalance
  - Median degree might be less than 10? Most vertices have few neighbors



(a) Twitter In-Degree

POWERGRAPH

Programming Model:
Gather-Apply-Scatter

→ extends think-like-a-vertex

Sync / Async execution

→ single machine

Better Graph Partitioning
with vertex cuts

→ Distributed

Delta based filtering to reduce work in next iteration!

GATHER-APPLY-SCATTER

Gather: Accumulate info from nbrs

final message into apply

Apply: Accumulated value to vertex

Scatter: Update adjacent edges

Gather → input vertex, edge states
returns accumulator

Apply → vertex, accumulator
↳ update state

Scatter → updated vertex & can update nbr

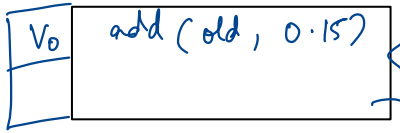
```
// gather_nbrs: IN_NBRS
gather(Du, D(u,v), Dv):
    return Dv.rank / #outNbrs(v)

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
```

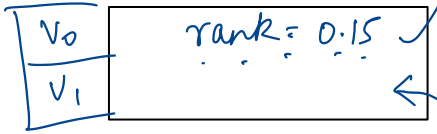
state of vertex u
state of edge (u,v)
state of v
Accumulators
vertex
messages
how much

EXECUTION MODEL

At beginning
→ Activate all vertices

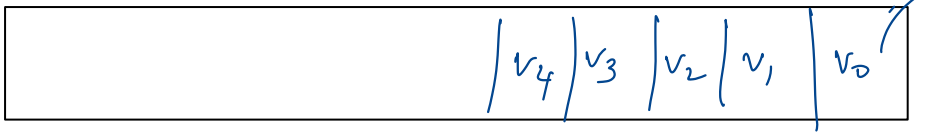
Accumulators



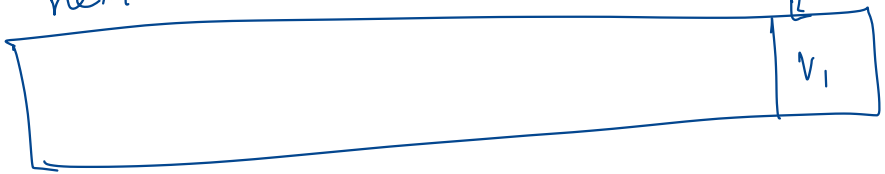


Vertex State

Active Queue



next iter



Gather

gather (v_0)

gather (v_1)

Apply

apply (v_0)

read acc

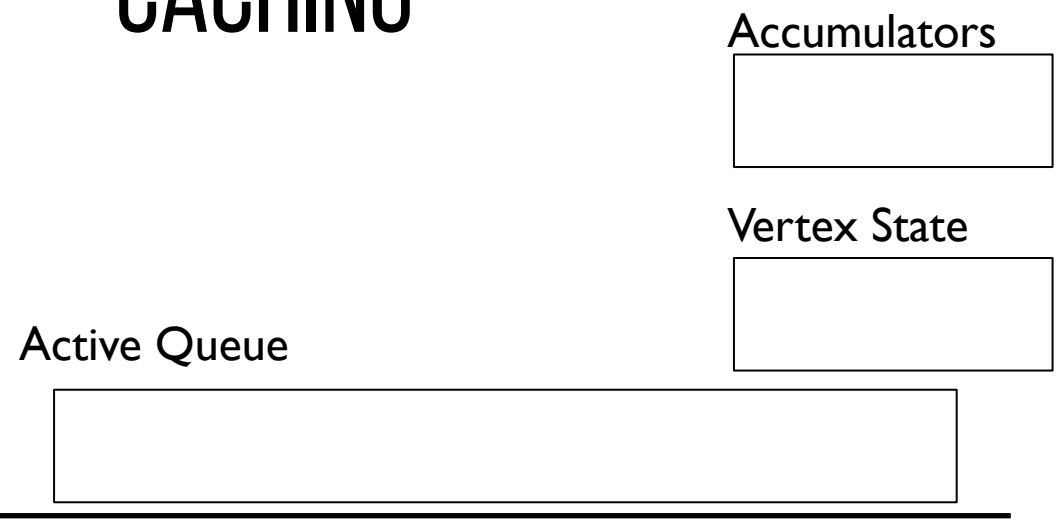
write back updated state

Scatter

check delta value

Activate (v_1)

CACHING



Delta caching

Cache accumulator value for vertex → Reuse acc computed in prev iteration → Skip gather phase for that vertex

Optionally scatter returns a delta

Accumulate deltas → If change is minimal dont activate vertex

SYNC VS ASYNC

Doesn't guarantee you get same result !

Sync Execution

Gather for all active vertices,

followed by Apply, Scatter

Barrier after each minor-step

gather phase

each thread call gather on a vertex

All the reads (of acc /state) happen all updates have finished !

Async Execution

Execute active vertices,

as cores become available

No Barriers! Optionally serializable

many threads waiting ??

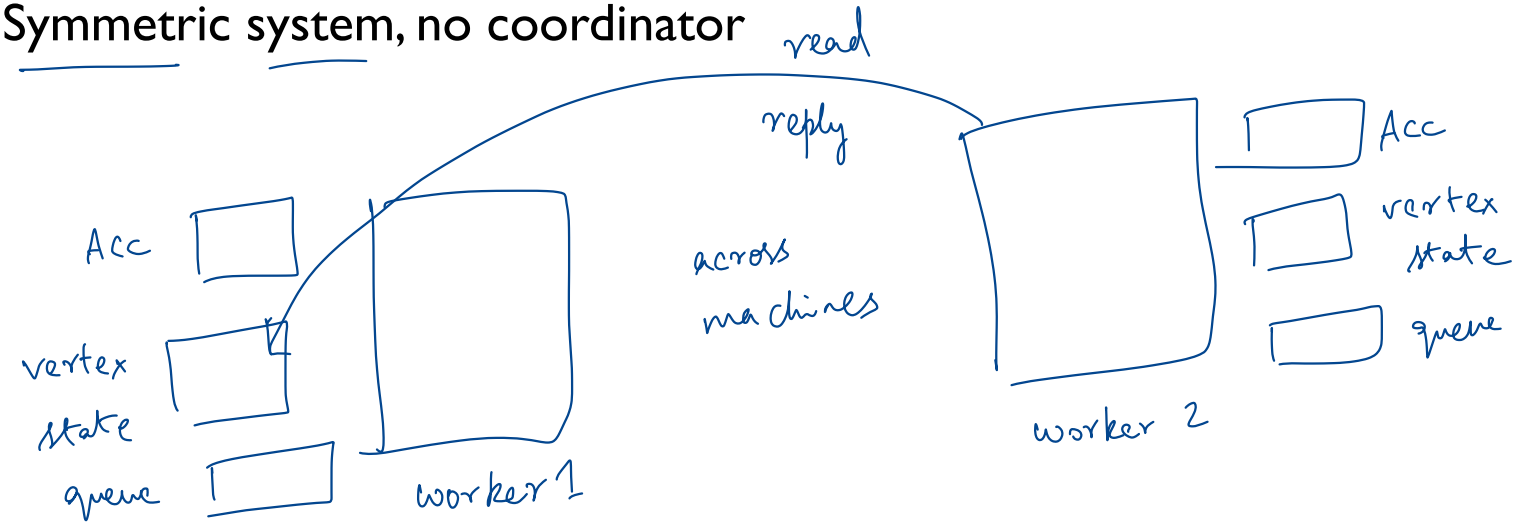
gather all vertices | apply all vertices | scatter all vertices

barrier

barrier

DISTRIBUTED EXECUTION

Symmetric system, no coordinator



Partition graph across machines

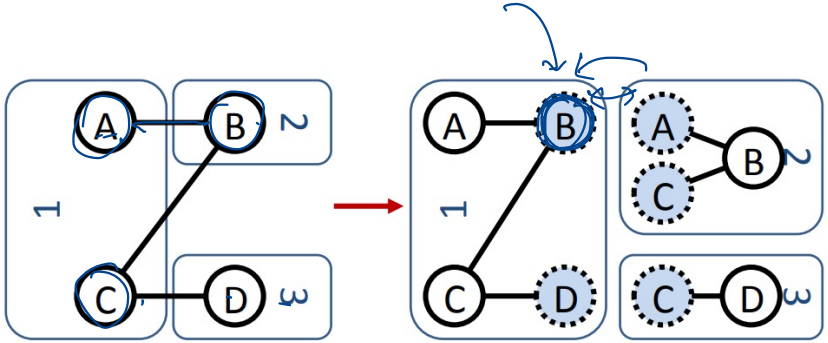
Communicate to spread updates, read state

"ghost" vertex

GRAPH PARTITIONING

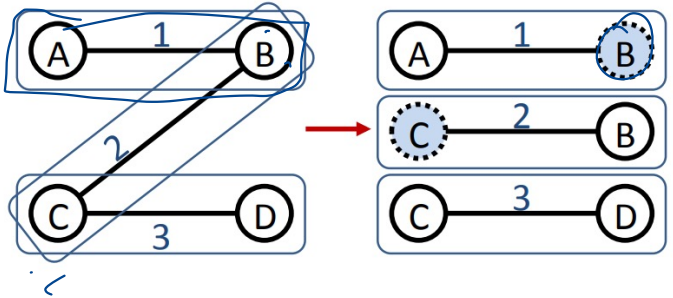
parallelize GAS programming model

replica



(a) Edge-Cut

- Assign a vertex to a machine
  - Edges are cut if other vertex is in a diff machine
- Imbalance for natural graphs.



(b) Vertex-Cut

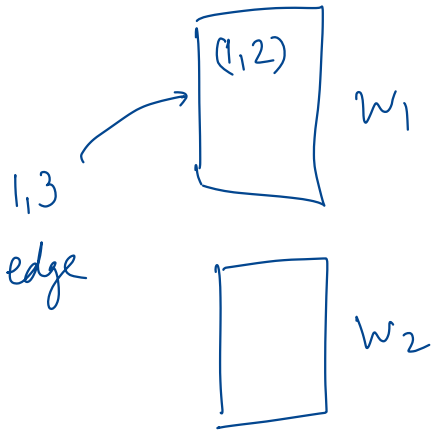
- Assign an edge to a machine
- One machine is primary for vertex and others replica

RANDOM, GREEDY OBLIVIOUS

Three distributed approaches:

Random Placement → Fast

↳ stream thru edges and place edge on random machine



Coordinated Greedy Placement

↳ if either vertex is placed favor that machine

Oblivious Greedy Placement

↳ Avoid coordination tracking if vertex is 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/co1BV6SzH7t31phGA

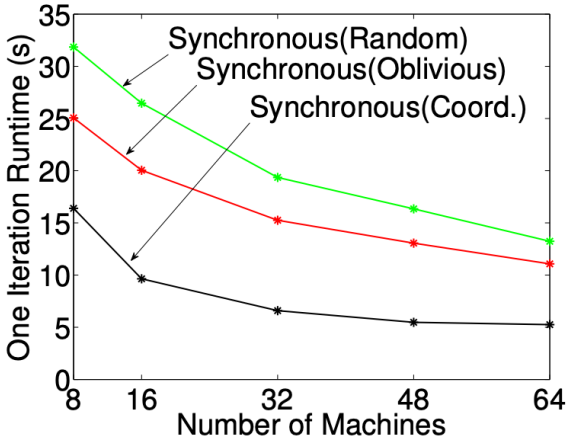
Consider the PageRank implementation in Spark vs synchronous PageRank in PowerGraph. What are some reasons why PowerGraph might be faster?

→ Better balance
  ↳ smarter partitioning & skew friendly processing vs. random partitioning

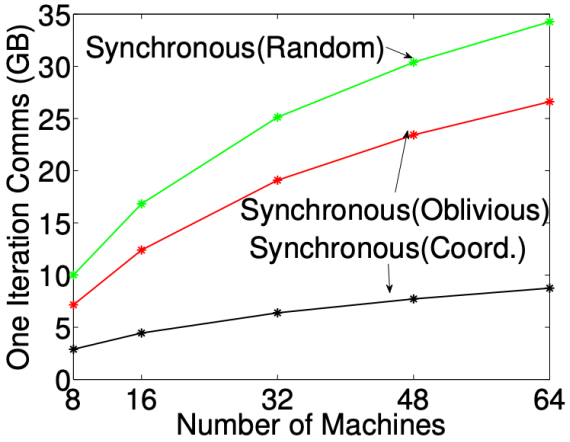
→ Lower communication
  ↳ faster than shuffle based approach in Spark

→ Scheduling granularity → skip vertices if not activated

→ Prog. model has features like caching accumulators reduce computation



(a) Twitter PageRank Runtime



(b) Twitter PageRank Comms

NEXT STEPS

Next class: Marius