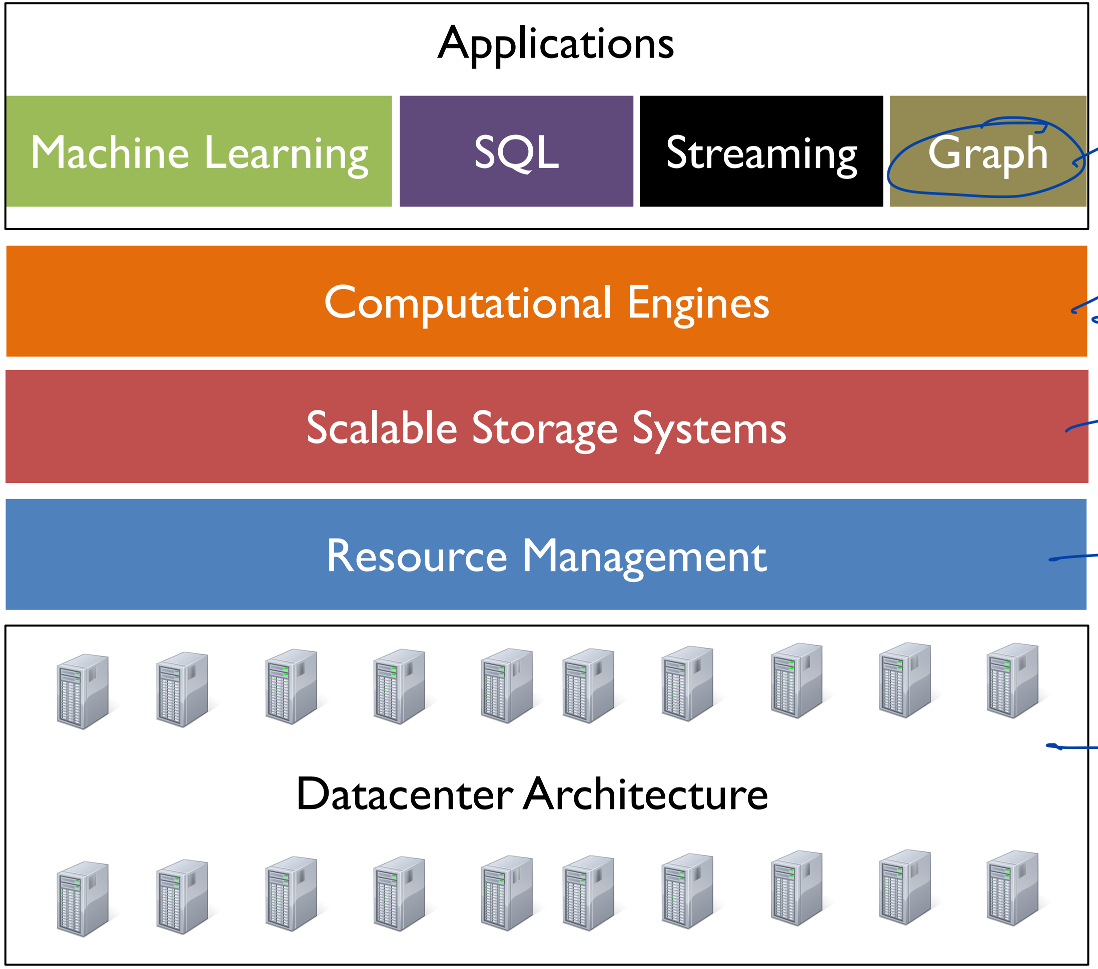
CS 744: POWERGRAPH

Shivaram Venkataraman

Fall 2019

ADMINISTRIVIA

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



Applications

Machine Learning

SQL

Streaming

Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture

MapReduce

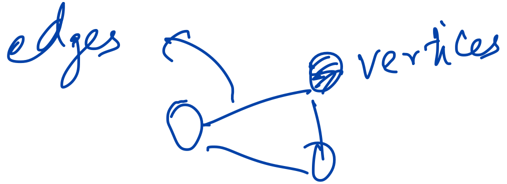
Spark

GFS

Mesos

DRF

GRAPH DATA



Datasets

Friendship among users

Knowledge base graph

- Entity : Berlin
  ↓ captial of
  Germany
-

Microservices / graph of systems

Map → locations & streets joining

Internet → hosts & links
IP

Application

Recommend new friends

Question answering

Debugging / Root cause analysis

Routing

"] → Routing of Internet Packets

GRAPH ANALYTICS

→ SQL

↳ Analytics on Tabular data

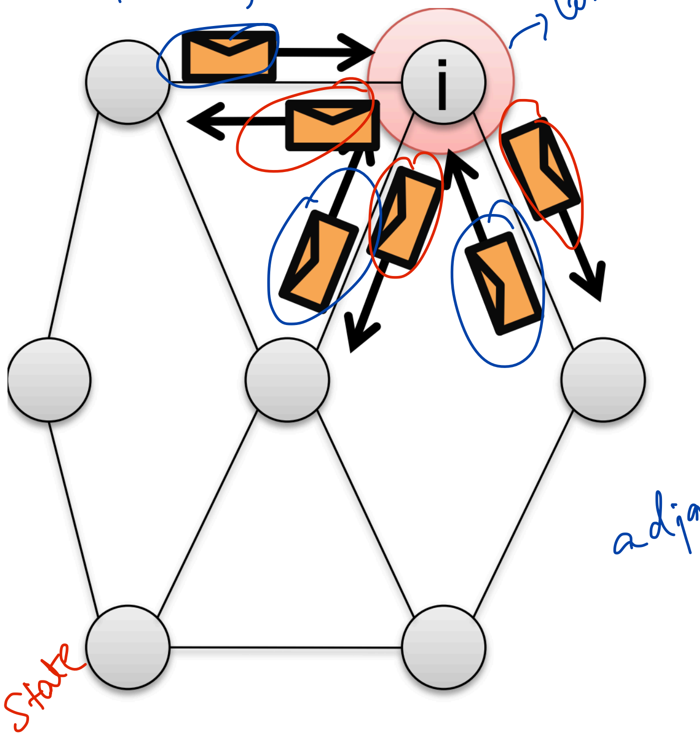
Perform computations on graph-structured data

Examples

- PageRank
- Shortest path
- Connected components
- …

PREGEL: PROGRAMMING MODEL

rdd - map (
// what I want to do for a row)



recv for incoming

```
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);
```

send outcoming

adjacency list

Node	Neighboring

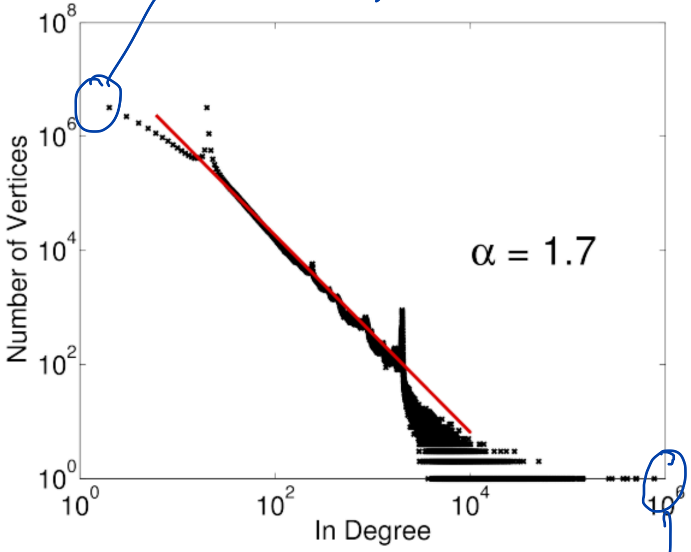
- Vertices have state only
- Neighboring vertices can communicate

NATURAL GRAPHS

- Degree distribution is very skewed

- Imbalance
  - Computation
  - Messages / Communication
  - State $\propto$ num of neighbors

1 million vertices degree ≈ 1



(a) Twitter In-Degree

1 million

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

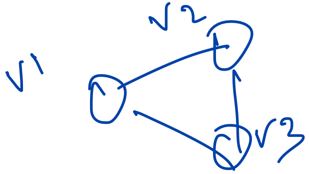
Edge state , vertex state

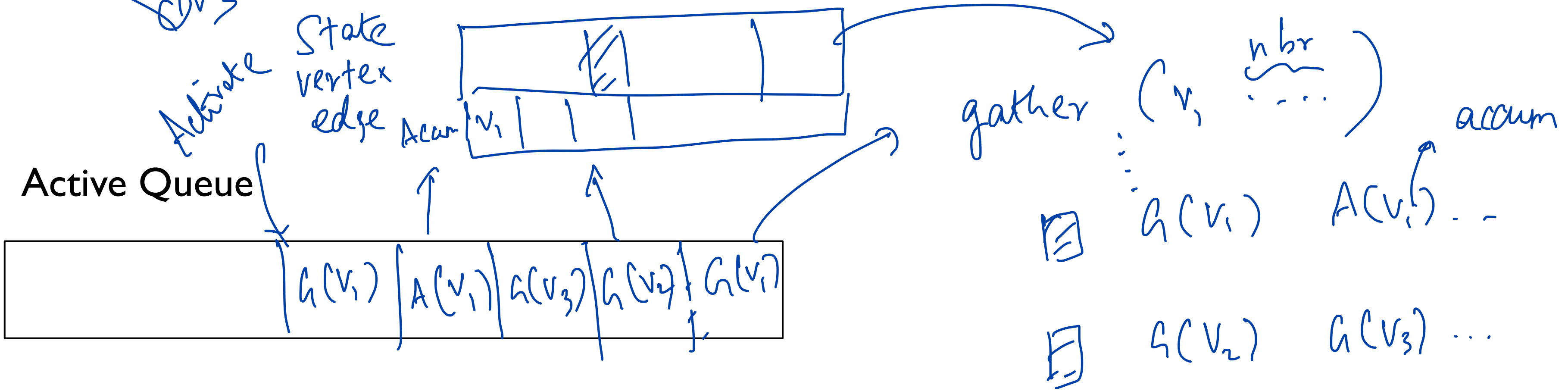
```
// gather_nbrs: IN_NBRS
gather(Du, D(u,v), Dv):
    return Dv.rank / #outNbrs(v)

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
```

Accumulator

Associative

EXECUTION MODEL, CACHING





Delta caching

Cache accumulator value for vertex

Optionally scatter returns a delta

Accumulate deltas

→ Optimization to reduce number of nbrs gather on

Consistency

- Visible to gather?

VS
ES
memory
Queue

SYNC VS ASYNC

Sync Execution

Gather for all active vertices,

followed by Apply, Scatter

Barrier after each minor-step

Structured

next iteration

G(V1)

G(V2)

⋮

Barrier

A(Vi)

change state

⋮

Barrier

S(V1)

⋮

Async Execution

Execute active vertices,

as cores become available

No Barriers! Optionally serializable

G(V1)

A(V1)

G(V2)

⋮

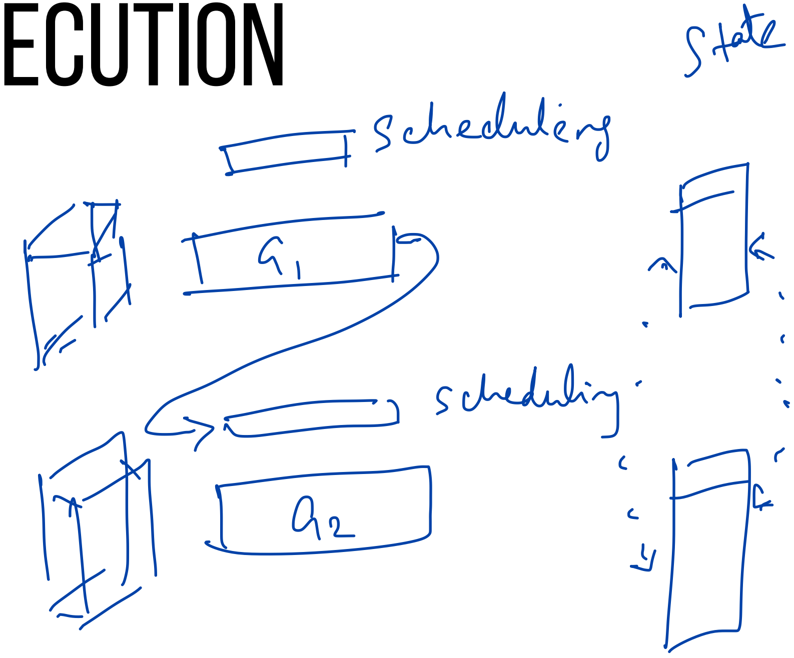
Not all vertices see same state

DISTRIBUTED EXECUTION

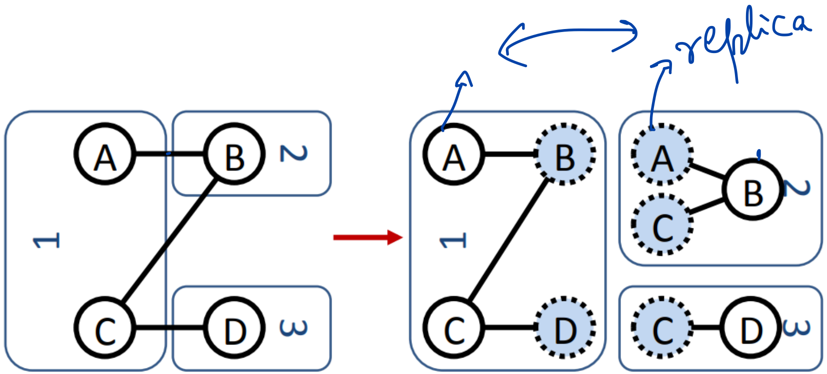
Symmetric system, no coordinator

Load graph into each machine

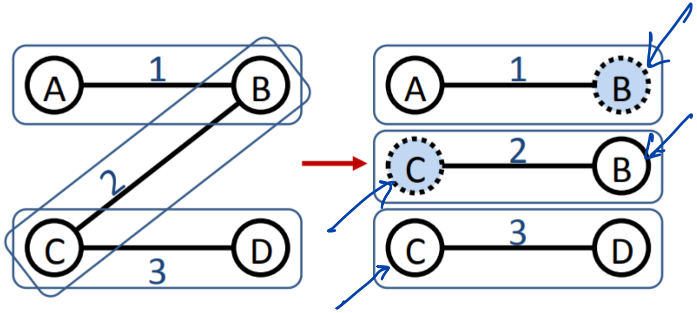
Communicate across machines to spread updates, read state



GRAPH PARTITIONING



(a) Edge-Cut



(b) Vertex-Cut

No split in computation

→ vertex is in 1 machine

when split edge

→ track vertex state across split

Vertex has a lots of edges, imbalance

Edge is in 1 machine

Vertices can be split across machines

RANDOM, GREEDY OBLIVIOUS

Data loading

Three distributed approaches:

Number of replicas of a vertex

Random Placement

↳ stream place edges on machines

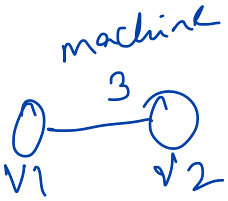
fast data loading

Coordinated Greedy Placement

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

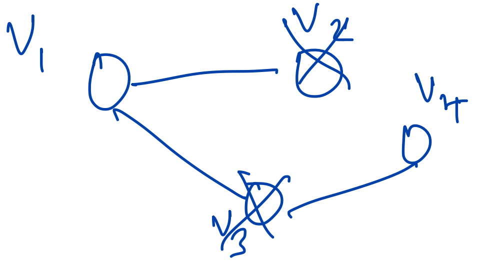
Oblivious Greedy Placement

Avoid this synchronization while data loading



V	mcs
V_1	1,3,5
V_2	3,4

OTHER FEATURES



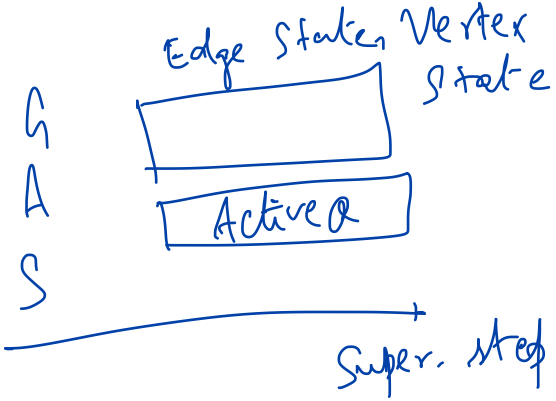
Async Serializable engine

- Preventing adjacent vertex from running simultaneously
- Acquire locks for all adjacent vertices

A(V1) G(V4)

Fault Tolerance

- Checkpoint at the end of super-step for sync
- For Async?

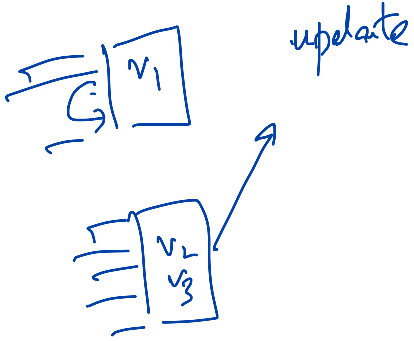


DISCUSSION

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

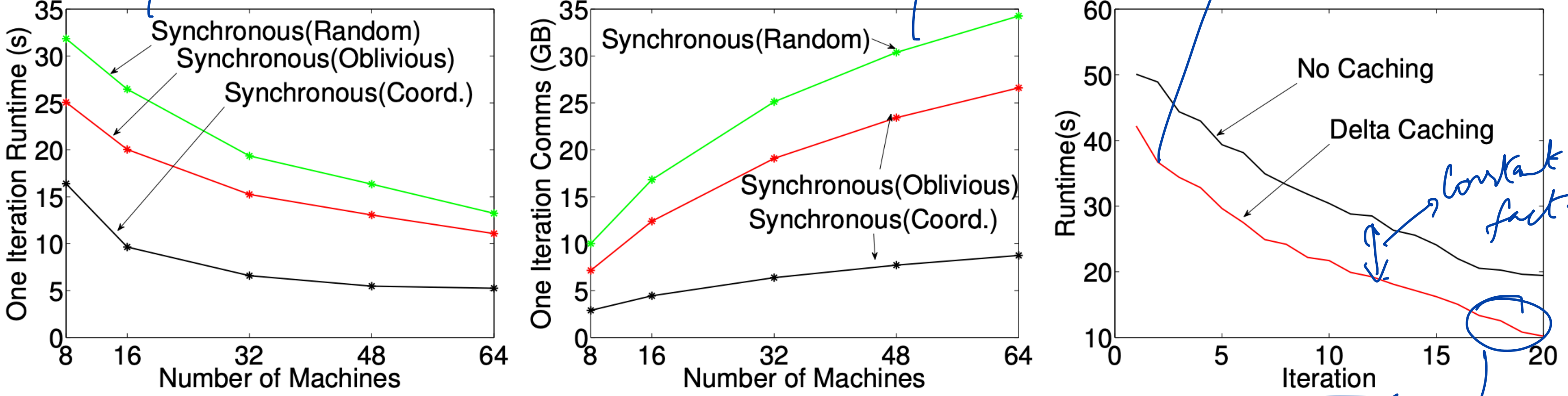
Partitioning function in Spark ?

random is slower

= Suboptimal split | vertices replicated too many?

random has more comm

Delta caching improve perf



constant factor..

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

Activation ↓ vertex → Comp/ gather

Activation based on delta

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 STEPS

Next class: GraphX

Sign up for project check-ins!