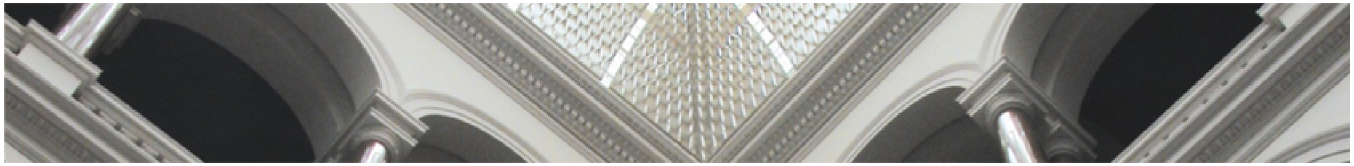
Technische Universität Berlin

Specialized Systems for Large-Scale Learning

Sebastian Schelter | Database Group, TU Berlin | Scalable Data Science: Systems and Methods

*with slides from Joseph Gonzalez and Carlos Guestrin



Overview

- Vertex-Centric Graph Processing Systems
- Graph-Based Machine Learning Systems
- Parameter Servers
- Summary

Overview

- **Vertex-Centric Graph Processing Systems**
- Graph-Based Machine Learning Systems
- Parameter Servers
- Summary

Graph Processing in MapReduce

- **unnecessarily slow**: each iteration is a single MapReduce job (or a series of jobs) with lots of overhead
  - separately scheduled
  - graph structure is read from disk
  - the intermediary result is written to HDFS

- **hard to implement**: a join has to be implemented by hand, lots of work, best strategy is data dependent

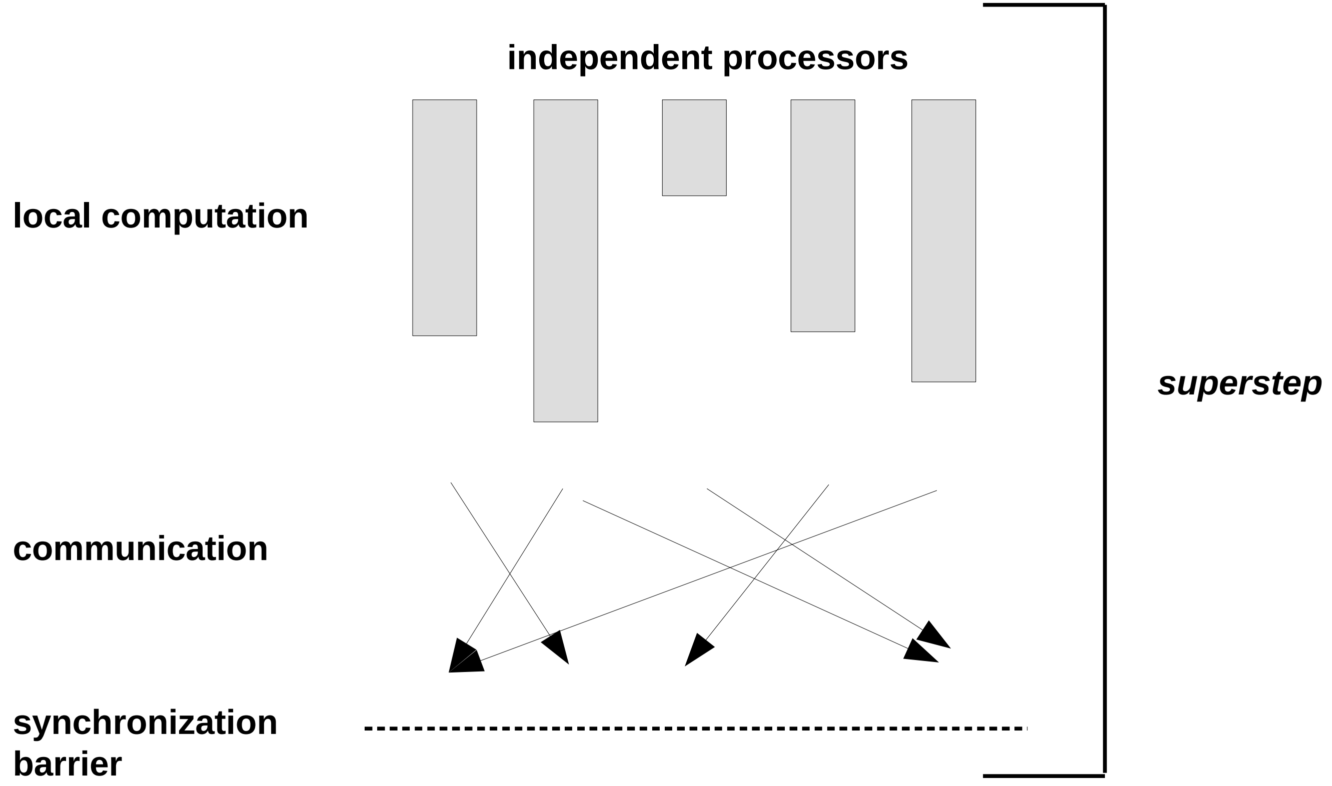
Vertex-Centric Graph Processing

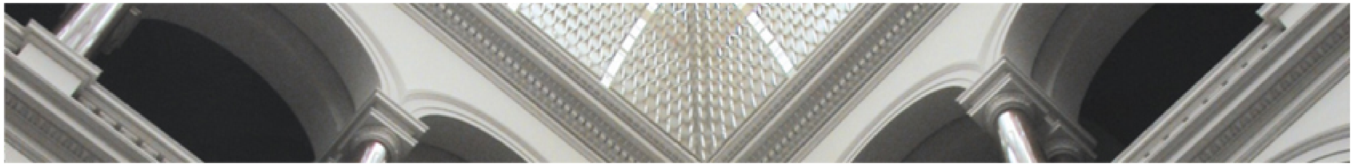
- core abstraction of the distributed graph processing system **Google Pregel**
- open source implementation available: **Apache Giraph**
- **specialized system, not a general dataflow system**

- Pregel defines **computational model for distributed graph processing**:
  - programs consist of iterations where vertices get messages from the previous iteration, modify their own state and send messages to other vertices

  - heavily inspired by **'Bulk Synchronous Processing' (BSP)**, a general model for parallel computations

Bulk Synchronous Parallel (BSP)





Technische Universität Berlin

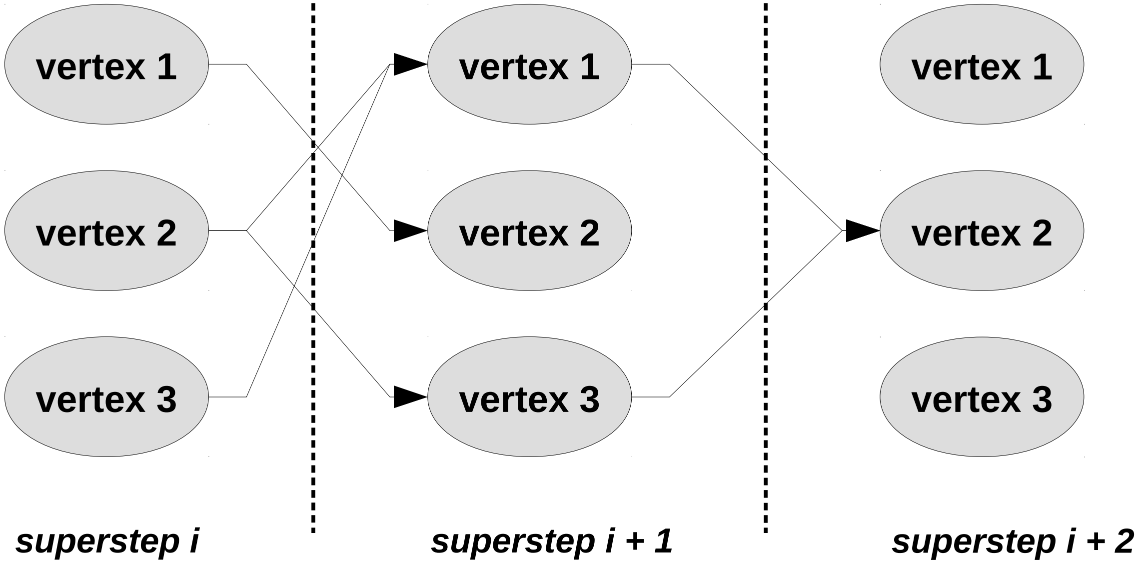
Vertex-Centric Graph Processing

- analogous to BSP, iterations are called supersteps in vertex-centric graph processing

- **anatomy of a superstep**
  - system invokes UDF for every vertex in parallel
  - UDF defines behavior of vertex *v* in superstep *s*
    - receives messages assigned to it in superstep *s-1*
    - can modify state of *v* (as well as graph topology)
    - can send messages to other vertices, which will be delivered in superstep *s+1*
  - execution synchronized via **global superstep barrier**

Vertex-Centric BSP: “think like a vertex”

- each vertex has an **id**, a **value**, the **adjacent neighbor ids** and the corresponding **edge values**
- each vertex is **invoked in each superstep**, can **re-compute its value** and **send messages to other vertices**, which are **delivered over superstep barriers**
- advanced features : **termination votes, combiners, aggregators, topology mutations**

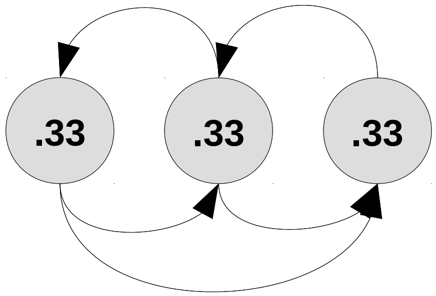


Vertex-Centric Graph Processing

- simple **PageRank** implementation using vertex-centric graph processing

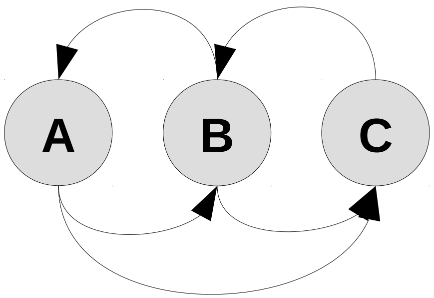
```
class PageRankVertex
  compute(messages):
    rank = 0.85 * sum(messages) + 0.15 * total_num_vertices()
    set_state(rank)

    if get_superstep() < 20 :
      send_message_to_all_neighbors( rank / get_num_neighbors() )
    else:
      vote_to_halt()
```

Example: PageRank

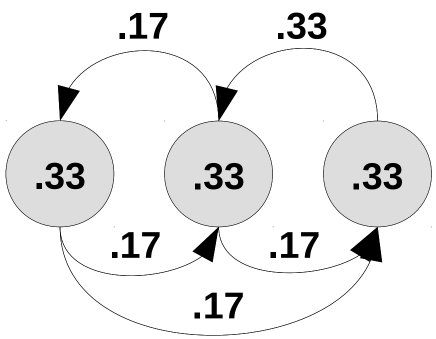


superstep 0

input graph

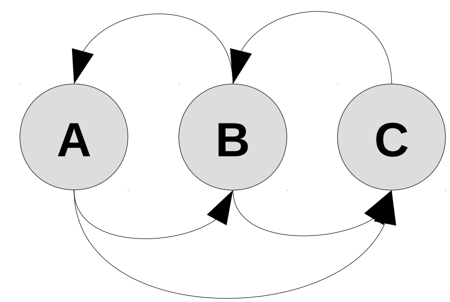


Example: PageRank

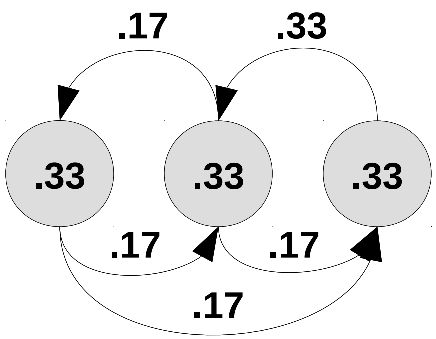


superstep 0

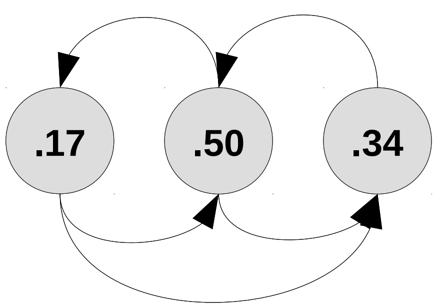
input graph



Example: PageRank

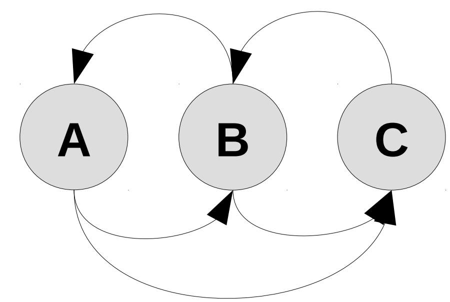


superstep 0

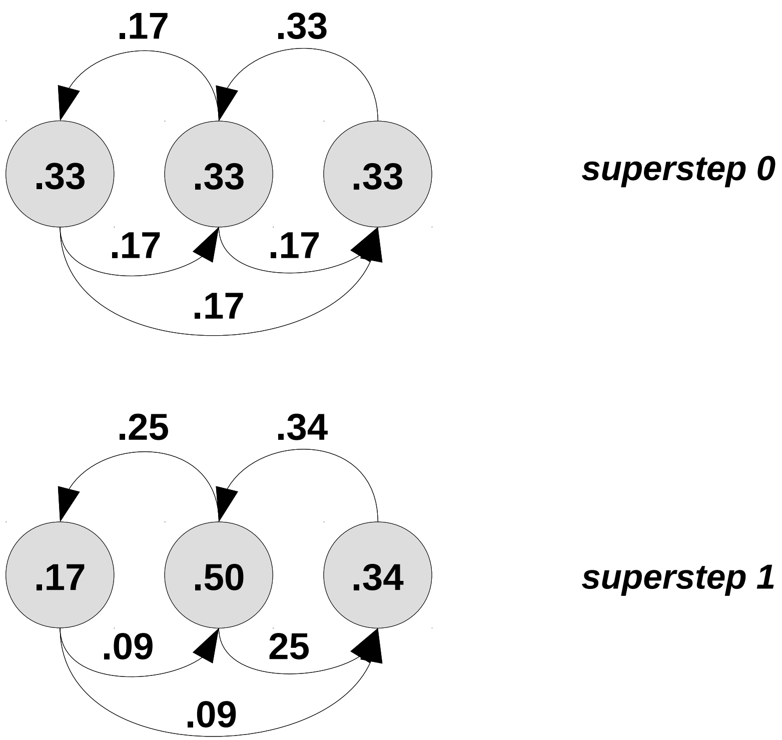


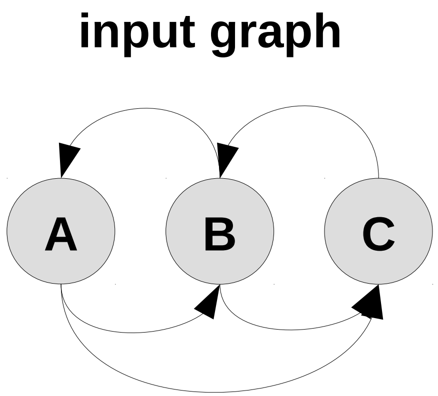
superstep 1

input graph



Example: PageRank





Architecture

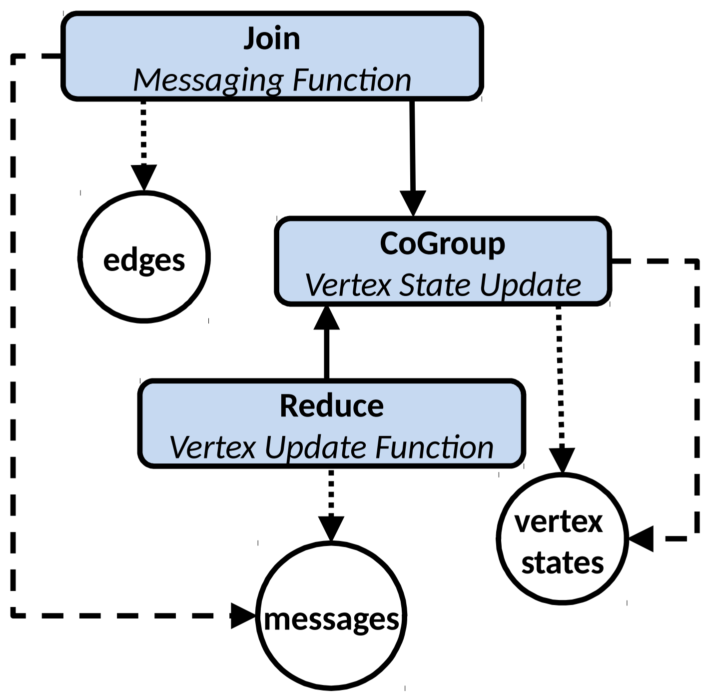
- Pregel uses **master-slave architecture**

- initialization
  - **graph is partitioned by vertex id**, such that a vertex and its whole adjacency list live in the same partition
  - master assigns partitions to workers which load graph in memory

- **master coordinates the supersteps**
  - during a superstep, **workers invoke UDF on their local partitions** and asynchronously deliver messages
  - **synchronization barrier** via distributed locking service
  - execution continues as long as there are active vertices or messages to be delivered
  - during termination, vertices output their state as result of the computation

Vertex-Centric Graph Processing (4)

- general data flow systems **efficiently emulate vertex-centric graph processing systems**

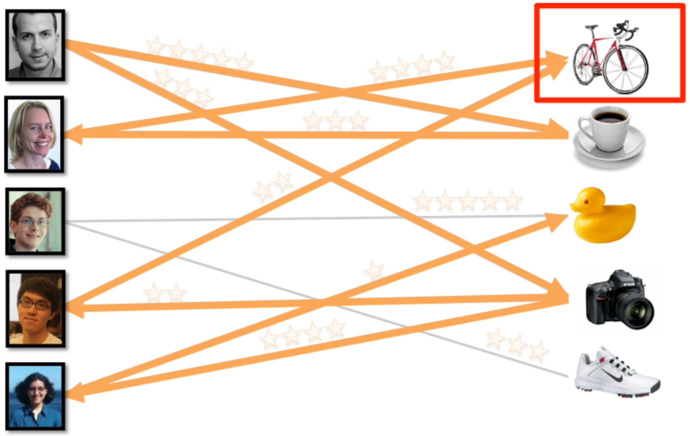


Overview

- Vertex-Centric Graph Processing Systems
- **Graph-Based Machine Learning Systems**
- Parameter Servers
- Summary

Graph-based Machine Learning

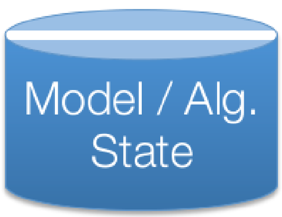
- the **data dependencies** of many machine learning problems can be viewed as **a graph**
- e.g., **users and items in collaborative filtering**, probabilistic graphical models, ...

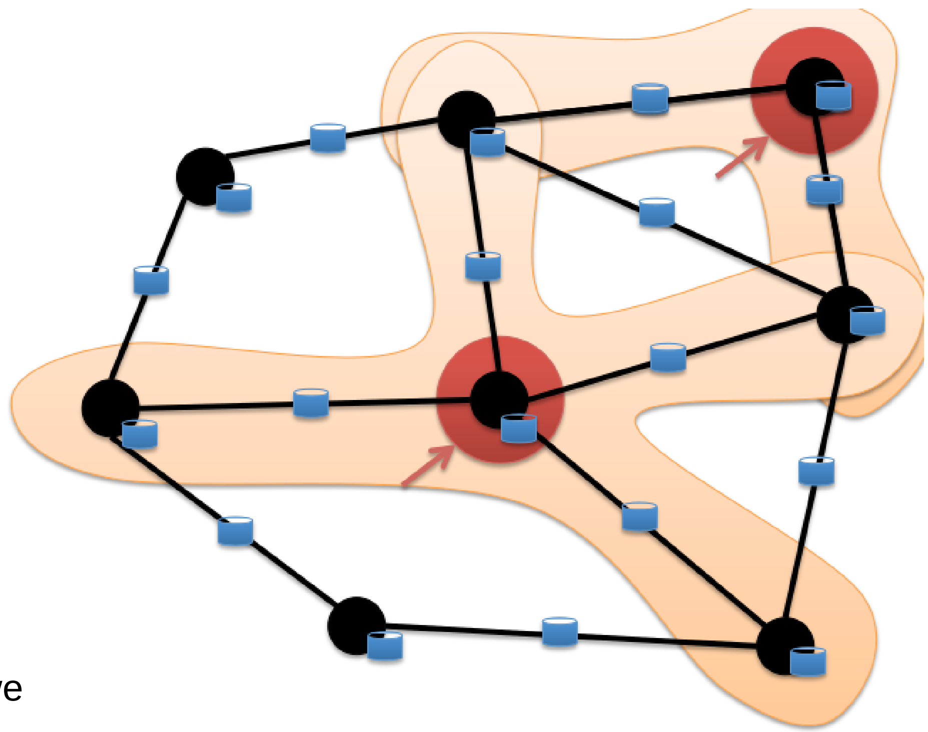


→ we can use **vertex centric graph processing for machine learning!**

Graph-Parallel Algorithms

- computations only depends on the neighbors of a vertex!





- restricted form of vertex-centric graph processing

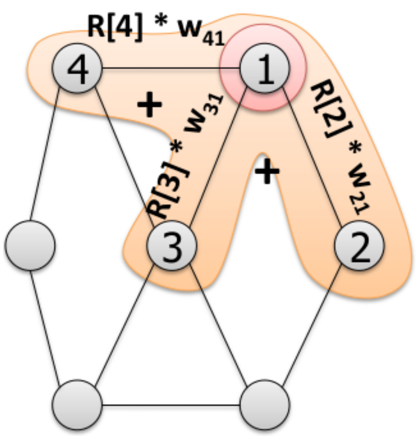
- question: is BSP the best way to execute these programs when we tackle machine learning problems?

GraphLab: Asynchrony & Shared State

- vertex-centric graph processing, but:
  - **shared state abstraction**: vertices **directly access** adjacent vertices and edges
  - **asynchronous execution** of vertex update programs

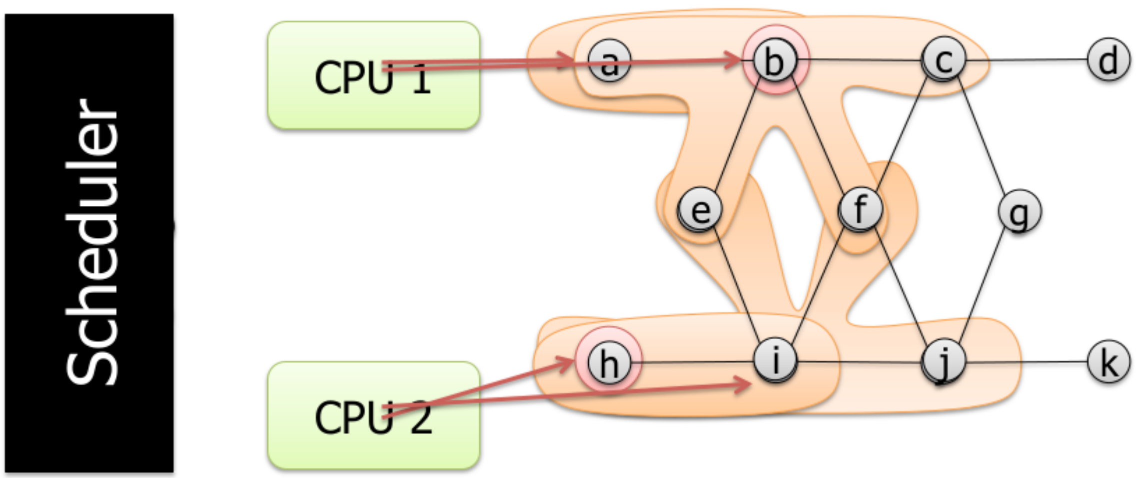
```
GraphLab_PageRank(i):
  // compute sum over neighbors
  total = 0
  foreach (j in neighbors(i)):
    total += rank[j] * w_ij
  // update the pagerank
  rank[i]  = 0.15 + total
  // trigger neighbors to run again
  if rank[i] not converged then
    signal neighborsOf(i) to be recomputed
```

signaled vertices are recomputed eventually



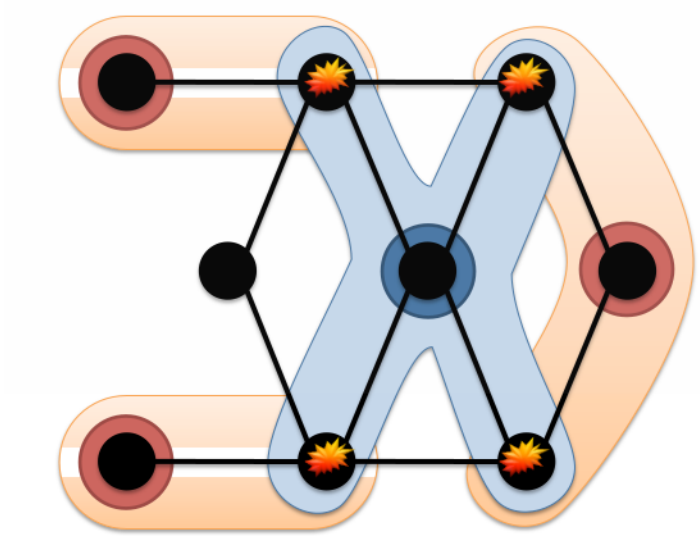
Scheduling of Vertex Updates

- scheduler determines which vertices are updated concurrently



Handling race-conditions

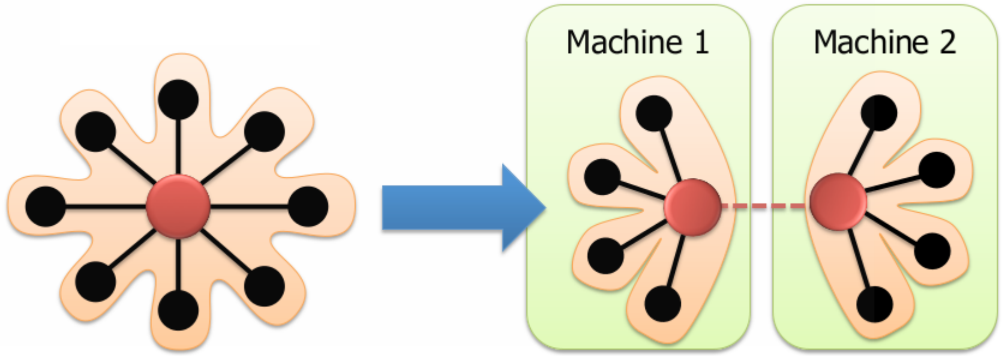
- GraphLab provides **different consistency levels** while scheduling vertex updates (different algorithms can live with different levels of consistency)
- can guarantee **sequential consistency**: for each parallel execution, there exists a sequential execution of update functions which produces the same result



Handling high-degree vertices

- many real world graphs have **highly skewed degree distributions**
  - **high degree vertices** make locking difficult and require high amount of communication

- countermeasures in Graphlab
  - **Gather – Apply – Scatter** paradigm to allow for pre-aggregation in vertex updates
  - **two-dimensional partitioning** of the data graph ("**vertex cut**")



Synchronous vs Asynchronous Graph Processing

Synchronous (BSP)

- **computation in phases**
  - all vertices participate
  - all messages are sent
- **simple to build**
  - no race conditions, barrier guarantees consistency
  - simple fault tolerance
- **slow convergence** for many ML problems
- math equivalent: Jacobi iterations

Asynchronous (GraphLab)

- **vertices see latest information from neighbors**
- **hard to build**
  - race conditions all the time
  - fault tolerance more complex
  - termination detection
- **fast convergence** for many ML problems
- math equivalent: Gauss-Seidel iterations

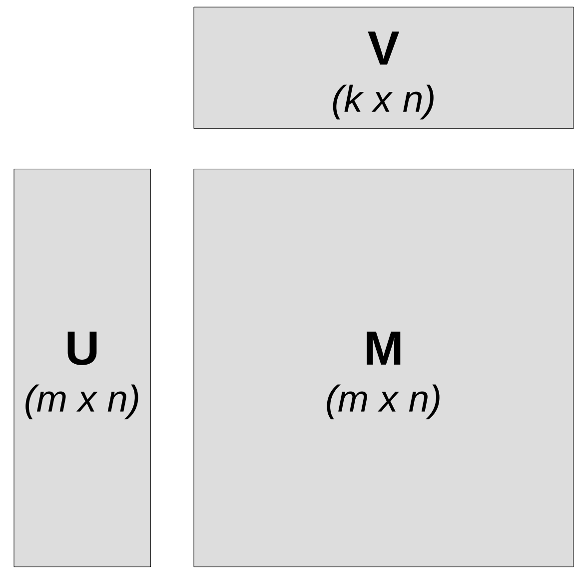
Overview

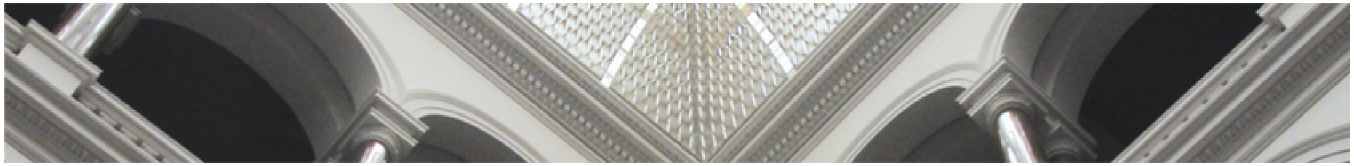
- Vertex-Centric Graph Processing Systems
- Graph-Based Machine Learning Systems
- **Parameter Servers**
- Summary

“Factorbird”: Scaling Matrix Factorization at Twitter

- **Recap: Latent Factor Models**
  - input data: sparse *m x n* **interaction matrix *M***
  - matrix factorization:
    - **find factor matrices *U* and *V***, such that *UV* approximates *M* and generalizes well to unseen parts of *M*
    - **related to Singular Value Decomposition**

- built a **parameter server for large-scale matrix factorization** during a **summer internship at Twitter**





Mathematical approach

- **standard approach:** minimize regularized squared error of predictions to observed data

$$\sum_{m_{ij}\ observed} (m_{ij} - u_i^T v_j)^2 + \lambda (\|u_i\|^2 + \|v_j\|^2)$$

- **extended model**
  - *g* **global bias**, b^U_i **bias of user *i***, b^V_j **bias of item *j***
  - *w(i,j)* **weight of prediction error** for interaction between user *i* and item *j*
  - *p(i,j)* **prediction function** for interaction strength between user *i* and item *j*

$$p(i,j) = g + b_i^U + b_j^V + u_i^T v_j$$

  - loss function

$$\frac{1}{2} \sum_{m_{ij}\ observed} w(i,j)(m_{ij} - p(i,j))^2 + \frac{\lambda}{2} (\|g\|^2 + \|b_i^U\|^2 + \|b_j^V\|^2 + \|U\|_F^2 + \|V\|_F^2)$$

Specifics

- **graph terminology**
  - we assume M represents a graph
  - i and j are vertices in this graph
  - observed entries of M are the edges of the graph
  - m_{ij} denotes weight of an edge in this graph
  - b^U and b^V are stored in U and V

- system should use **Stochastic Gradient Descent (SGD)** to learn the model
  - simple, fast convergence, easy to adapt to different models and loss functions

Focus & Challenges

- **system focus**

  - **ability to handle twitter-scale graphs** (scalability to large datasets more important than high performance on small datasets, matrices 'tall-and-wide')
  - **simple design, extendable to a streaming scenario**

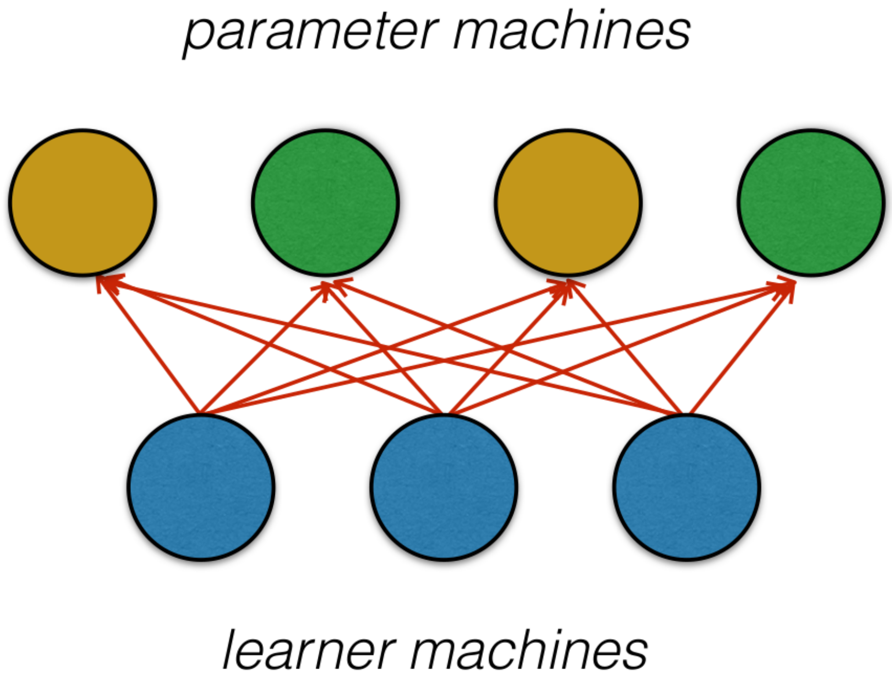
- **challenges**

  (1) model potentially larger than RAM on a single machine
  (e.g. U, V of rank 100 for 250M vertices ~ 200GB)

  (2) Conflicts occur when we run SGD in parallel
  (e.g. when two cores try to update the same u i in parallel, one update will be lost)

Handling Challenge (1)

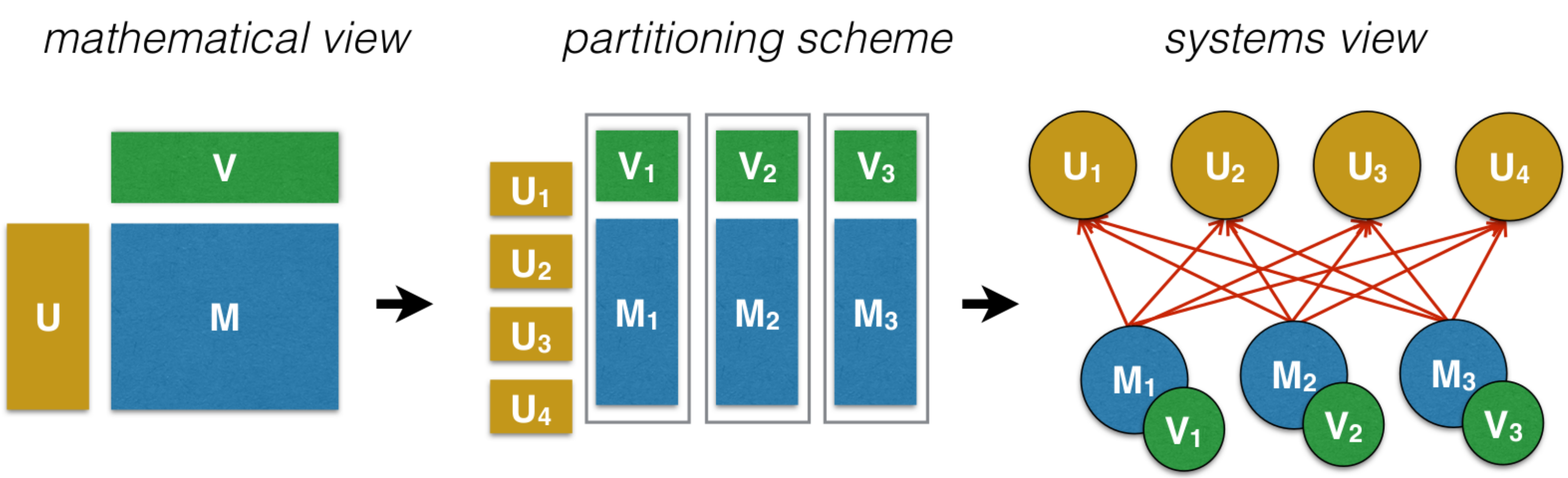
- use a **parameter server** architecture
  - partition model over a set of **parameter machines**
  - partition graph over a set of **learner machines**
  - learner machines fetch parameters, update them and write them back



Handling Challenge (2)

- **Hogwild!**
  - SGD can be implemented without any locking if most updates only modify small parts of the model
  - further optimization: one matrix can be co-partitioned with the inputs, updates to it will be local
    - (choice depends on whether the in-degree or out-degree distribution of the input graph is more skewed)

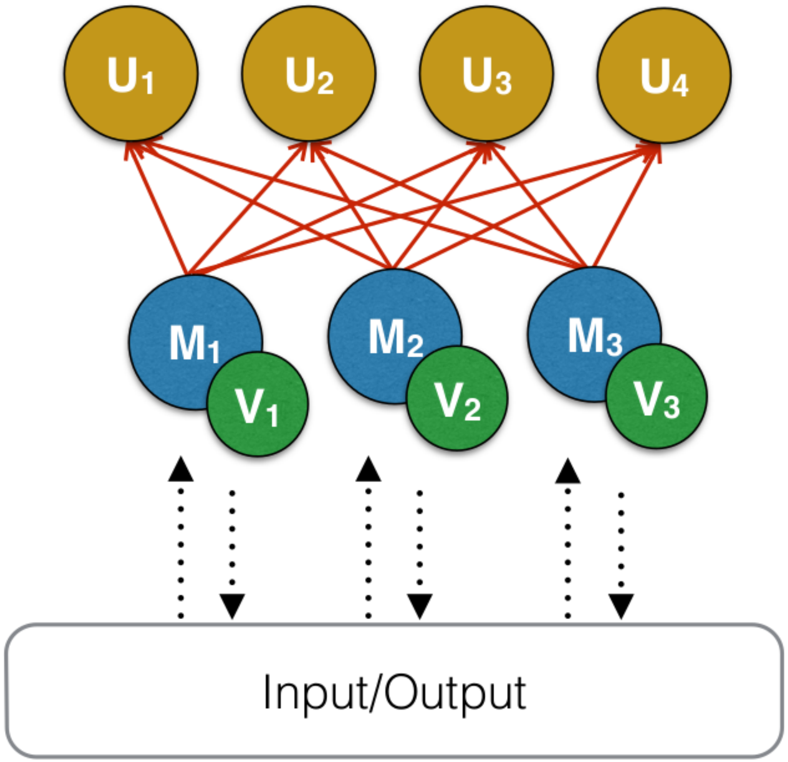
Big Picture



Co-partition M and V according to the number of learner machines

Co-locate partitions of M and V on learner machines

Architecture



Steps during execution

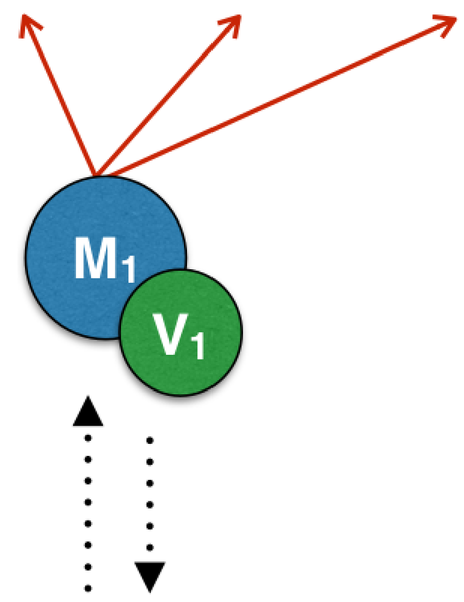
(1) load graph statistics

(2) instantiate local partition of V

(3) training:

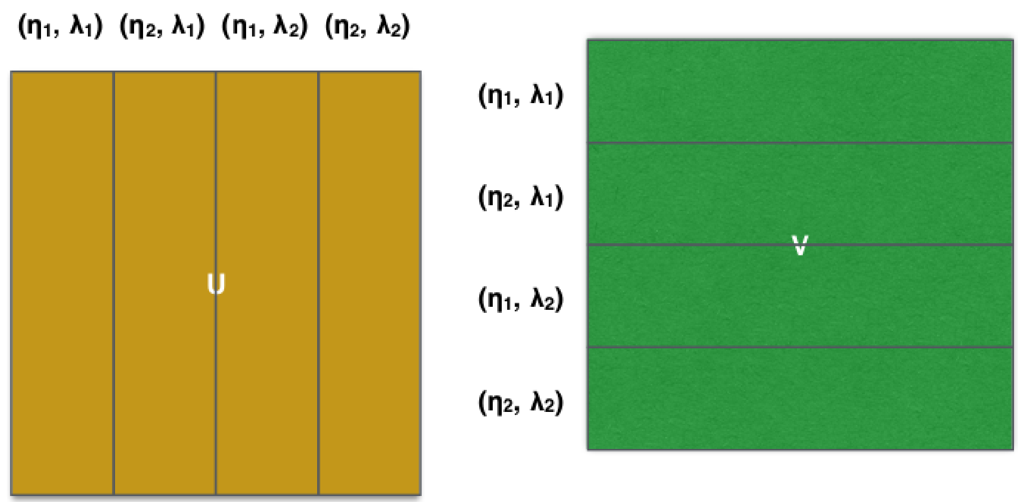
- stream edges
- fetch factor vectors
- update them with SGD,
- write vectors back to parameter machine

(4) save results to distributed filesystem

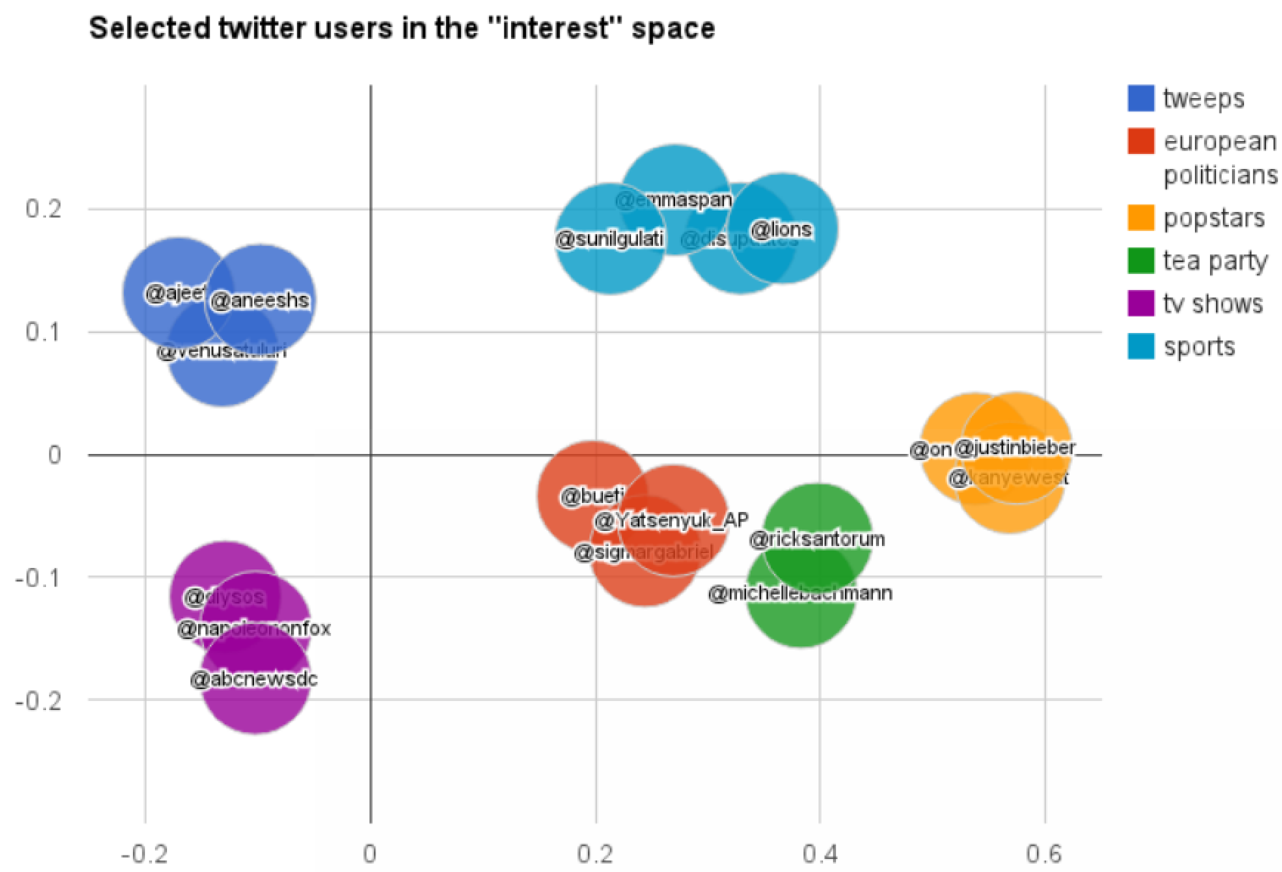


Model selection

- quality: prediction error on unseen data
  - problem: **heavily dependent on hyperparameters** (e.g., η (learning rate) and λ (regularization))
  - we don't know a way to analytically find well working hyperparameters
    → **grid search** over different hyperparameter combinations
  - **optimizing grid search**
    - single run per hyperparameter combination inefficient
      → learn many models at once, "multiplex" many different models into a large U & V



Results


Selected twitter users in the "interest" space
tweeps
european politicians
popstars
tea party
tv shows
sports
@sunilgulati
@lions
@aneeshs
@justinbieber
@Yatsenyuk_AP
@ricksantorum
@abcnewsdc
0.2
0.1
0
-0.1
-0.2
-0.2
0
0.2
0.4
0.6

Overview

Vertex-Centric Graph Processing Systems

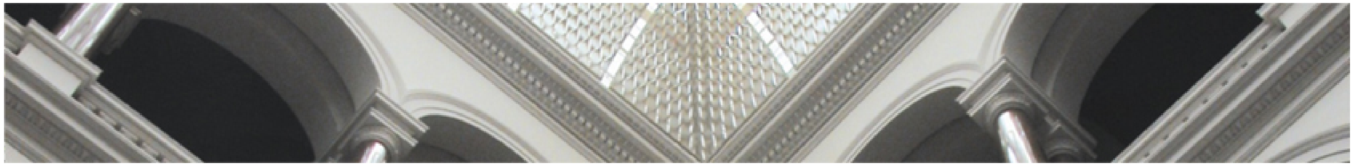
- Graph-Based Machine Learning Systems
- Parameter Servers
- **Summary**

Summary

- MapReduce **ill-suited for graph processing**
- Google Pregel introduces **vertex-centric BSP** as alternative
- **“think like a vertex”**: distributed graph processing based on vertex update functions and messaging

- **graph-based machine learning**
- adapts vertex centric paradigm, but adds **asynchrony for faster convergence** in many cases
- difficult to implement: race conditions, scheduling, consistency

- parameter servers option for **very large models**
- **Hogwild!** allows to run Stochastic Gradient Descent without locking
- hyperparameter optimization difficult in parameter server



Further Reading

- Valiant, L. G. (1990). *A bridging model for parallel computation.* Communications of the ACM, 33(8), 103-111.

- Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010, June). *Pregel: a system for large-scale graph processing.* In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (pp. 135-146). ACM.

- Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., & Hellerstein, J. M. (2012). *Distributed GraphLab: a framework for machine learning and data mining in the cloud.* Proceedings of the VLDB Endowment, 5(8), 716-727.

- Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., & Guestrin, C. (2012, October). *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs.* In OSDI (Vol. 12, No. 1, p. 2).

Further Reading

- Recht, B., Re, C., Wright, S., & Niu, F. (2011). *Hogwild: A lock-free approach to parallelizing stochastic gradient descent.* In Advances in Neural Information Processing Systems (pp. 693-701).

- Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., ... & Su, B. Y. (2014, August). *Scaling distributed machine learning with the parameter server.* In Proc. OSDI (pp. 583-598).

- Schelter, S., Satuluri, V., & Zadeh, R. (2014). *Factorbird-a Parameter Server Approach to Distributed Matrix Factorization.* arXiv preprint arXiv:1411.0602.