Graph Data Management

Analysis and Optimization of Graph Data Frameworks

presented by Fynn Leitow

Overview

1) Introduction
    a) Motivation
    b) Application for big data
2) Choice of algorithms
3) Choice of frameworks
    a) Framework implementation
4) Framework analysis
    a) Performance comparison
    b) Optimization techniques
5) Conclusion & Criticism

Motivation

- How to implement graph analysis algorithms on huge graphs?
- How do they perform?
- How about parallel computing?

Application for big data

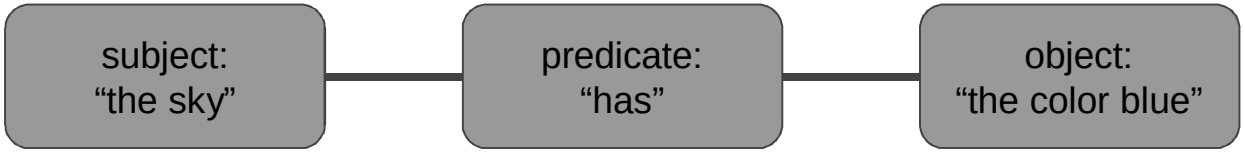
- Social Networks & Web of Data
- extremely large & dynamic
- can’t be handled by legacy programs

Social Networks

- vertices: people, pictures, videos
- edges: relations among nodes (friendship, follower)
- scale-free: follow power-law distribution (few vertices have high popularity)

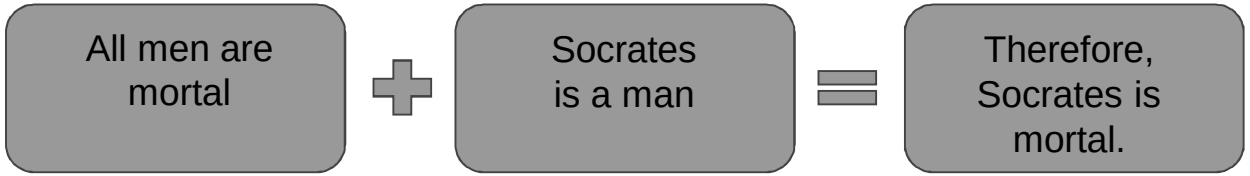
Web of Data

- Large-scale structured data by governments, researchers, companies
- Publishing principles:
  - Unique ressource identifier
  - publication at this URI in RDF triples (ressource-data-framework, statements similar to entity-relationship model but more general)
  - links to similar online ressources
- RDF example: “The sky has the color blue”



Typical queries

- social networks:
  - punctual updates (of vertices, adding edges)
  - transitive closures (“other people you might know”, ...)
  - betweenness queries (“common friends”, shortest path, ...)
- Web of data:
  - bulk inserts
  - joins
  - logical inference (deductions)



Objective of the paper

- Analyse existing graph frameworks - also for machine learning!
- native, hand-optimized implementation as reference point
- give suggestions to improve performance gap

Algorithms

PageRank (statistics)
Breadth-first search (graph traversal)
Triangle Counting (statistics)
Collaborative filtering (machine learning)

1.PageRank (site popularity)

➢how many links go to this this site?

➢technically: probability for a random walk to end on this vertex

$$PR^{t+1}(i) = r + (1 - r) * \sum_{j|(j,i)\in E} \frac{PR^t(j)}{\text{degree}(j)}$$

t - iteration

r - probability for random jump

e - set of directed edges

degree(j) - number of outgoing edges

2. Breadth-first search (BFS) - Graph traversal

- calculates “distance” (smallest number of edges) from start to any other vertex
- *Distance(start)* initialized to zero, all others to infinity
- iteratively computes for neighboring vertices:

$$Distance(i) = \min_{j|(j,i)\in E} Distance(j) + 1$$

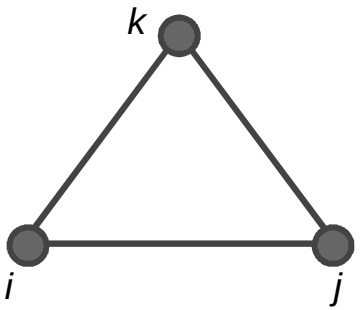
3. Triangle Counting - graph statistics

➢ Triangle := two vertices are both neighbors of a common third vertex

➢ Algorithm:

- each vertex shares his neighborhood list with its neighbors

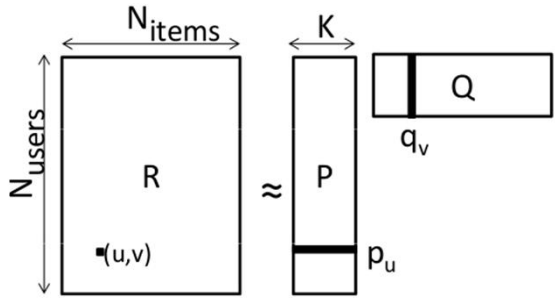
$$N_{triangles} = \sum_{i,j,k,i<j<k} E_{ij} \wedge E_{jk} \wedge E_{ik}$$



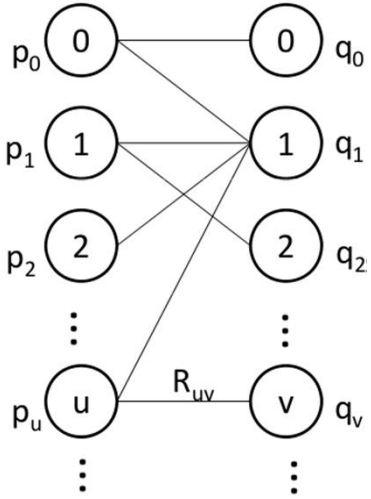
Collaborative filtering - machine learning

➢Recommendation system: predicts user ratings based on an incomplete set of (user, item) ratings

(a)Given a rating matrix R, find P and Q so that R = PQ is approximated best.



(a)



(b)

(b) find the bipartite graph with edge weights $R_{u,v}$

Overview

Algorithm	Application	Graph type	Vertex property	Message size (Bytes/edge)
PageRank	Graph statistics	Directed, unweighted edges	Double (*pagerank*)	Constant (*8*)
Breadth First Search	Graph traversal	Undirected, unweighted edges	Int (*distance*)	Constant (*4*)
Collaborative Filtering	Machine learning	Bipartite graph; Undirected, weighted edges	Array of Doubles (p_u or q_v)	Constant (*8K*)
Triangle Counting	Graph statistics	Directed, unweighted edges	Long (*# triangles*)	**Variable (*0-10^6*)**

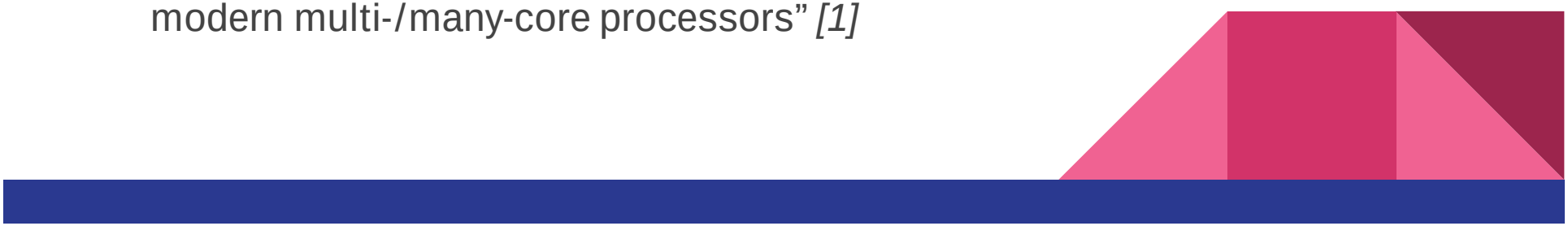
➢We focus on PageRank and BFS

Frameworks

Graphlab
CombinatorialBLAS
SocialLite
Galois
Giraph

Explanation: Framework & Ninja gap

- Framework: layered structure indicating what kind of programs can/should be built and how they should interrelate
- can include programs, specify interfaces, offer programming tools,...
- Ninja performance gap: “Performance gap between naively written C++/C code that is parallelism unaware (often serial) and best-optimized code on modern multi-/many-core processors” *[1]*



Choice of Frameworks:

(Stanford University)

Galois

(Austin Texas University)

GraphLab
Carnegie Mellon

APACHE
GIRAPH

COMBINATORIAL_BLAS
(UCSB)

Overview

Framework	Programming model	Language	Graph Partitioning	Communication layer
GraphLab	Vertex	C++	1-D (vertex-part.)	Sockets
CombBLAS	Sparse (adjacency) matrix	C++	**2-D (edge-part.)**	**MPI**
SociaLite	Datalog (declarative, deductive database tables)	Java	1-D	Sockets
Galois	Task-based	C/C++	N/A	N/A
Giraph (Hadoop)	Vertex	Java	1-D	Netty

- Galois: parallel computing framework for irregular data structures

PageRank - vertex programming

- Graphlab & Giraph, similar for Galois
- runs on a single vertex and communicates with adjacent vertices
- $PR \longleftarrow r$

  for $msg \in$ incoming messages **do**

  $\quad PR \longleftarrow PR + (1 - r) * msg$

  Send $\frac{PR}{degree}$ to all outgoing edges

PageRank - sparse matrix (CombBLAS)

➢ single iteration of PageRank:

$$\mathbf{p}_{t+1} = r\mathbf{1} + (1-r)\mathbf{A}^T\tilde{\mathbf{p}}_t$$

A - adjacency matrix

p_t - vector of PageRank values

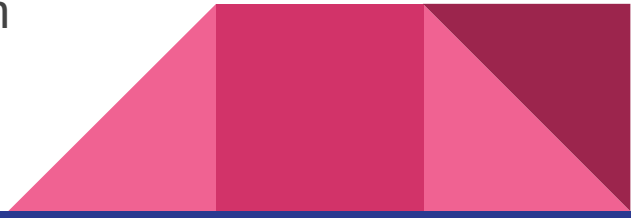
$\tilde{p}_t = p_t$ / [vector of vertex out-degrees]

PageRank - declarative (SociaLite)

$$\textsc{Rank}[n](t+1, \$\textsc{Sum}(v)) \text{ :- } v = r$$

$$\text{:- } \textsc{InEdge}[n](s), \textsc{Rank}[s](t, v_0), \textsc{OutDeg}[s](d), v = \frac{(1-r)v_0}{d}.$$

- Head: PageRank of vertex n for iteration *t+1* is sum of two rules:
- 1st: constant term, 2nd: normalized values from iteration *t*
- *InEdge[n](s)*: vertices in 1st, neighbors in 2nd column
- second version optimized for distributed machines



BFS - vertex programing (GraphLab, Giraph)

$$\textbf{for } msg \in \text{incoming messages } \textbf{do}$$
$$\quad Distance \longleftarrow \min(Distance, msg + 1)$$
$$\text{Send } Distance \text{ to all outgoing edges}$$

- Initialize *Distance* to zero or infinity, respectively
- Continue until there are no more updates

Breadth First Search - sparse matrix (CombBLAS)

- matrix-vector multiplication in each iteration:

$$v = A^T s$$

v - non-zeros indicate start vertices for next iteration

A - adjacency matrix

s - starting vertices

Breadth First Search - SociaLite

$$\mathrm{BFS}(t, \$\mathrm{MIN}(d)) \text{ :- } t = \mathrm{SRC}, d = 0$$
$$\text{ :- } \mathrm{BFS}(s, d_0), \mathrm{EDGE}(s, t), d = d_0 + 1.$$

➢ 1st rule handles source, 2nd recursively follows neighbors

Breadth First Search - Galois

```
Graph G
src.level = 0
worklist[0] = src
i ⟵ 0
while NOT worklist[i].empty() do
    foreach ( n : worklist[i] ) in parallel do
        for dst: G.neighbors( n ) do
            if dst.level == ∞ then
                dst.level ⟵ n.level + 1
                worklist[i+1].add(dst)
    i ⟵ i+1
```

- Work lists are maintained & parallel processed for each level by Galois

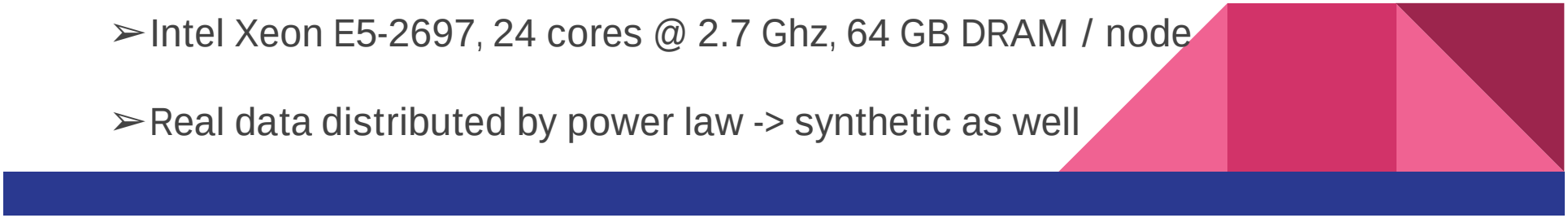
Framework Analysis

Datasets - Performance on single and multiple nodes

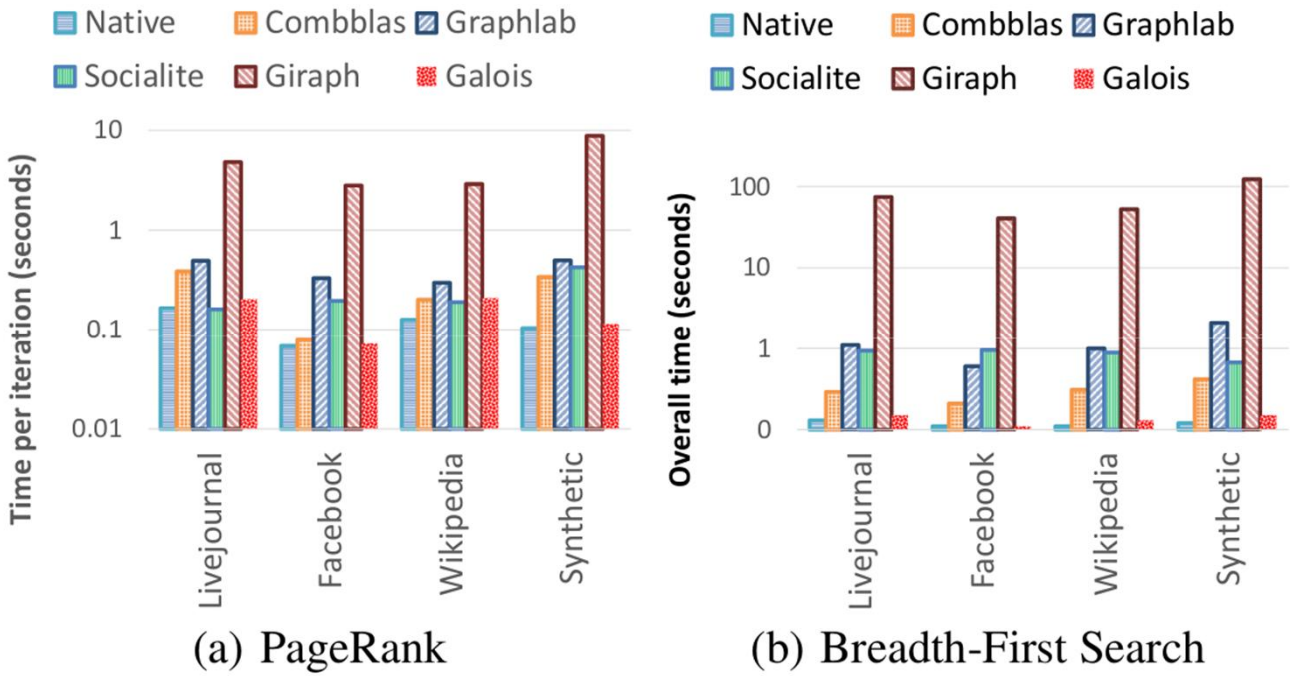
Experimental Setup

Dataset	FB, Wiki, Livejournal	Netflix	**Twitter**	**Yahoo Music**	Synthetic Graph500 (64 nodes)	Synthetic Collaborative Filtering (64 nodes)
# vertices ($k = 10^3$)	*3 - 5 M*	*480 k* users *18 k* movies	***62 M***	*1.0 M* users *0.6 M* items	*537 M*	*63 M* users *1.3 M* items
# edges ($M = 10^6$)	*42 - 85 M*	*99 M* ratings	***1468 M***	*253 M* ratings	*8539 M*	*16743 M* ratings

- Twitter & Yahoo large enough for multiple (4, 16 for triangle) nodes
- Intel Xeon E5-2697, 24 cores @ 2.7 Ghz, 64 GB DRAM / node
- Real data distributed by power law -> synthetic as well



Performance results (single node)

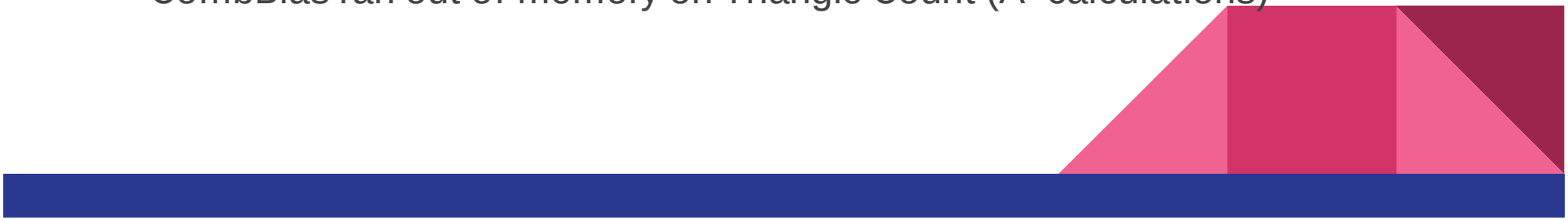


- native fastest
- Galois very fast (single node only)
- Giraph slow
- synthetic data in line with real-world results

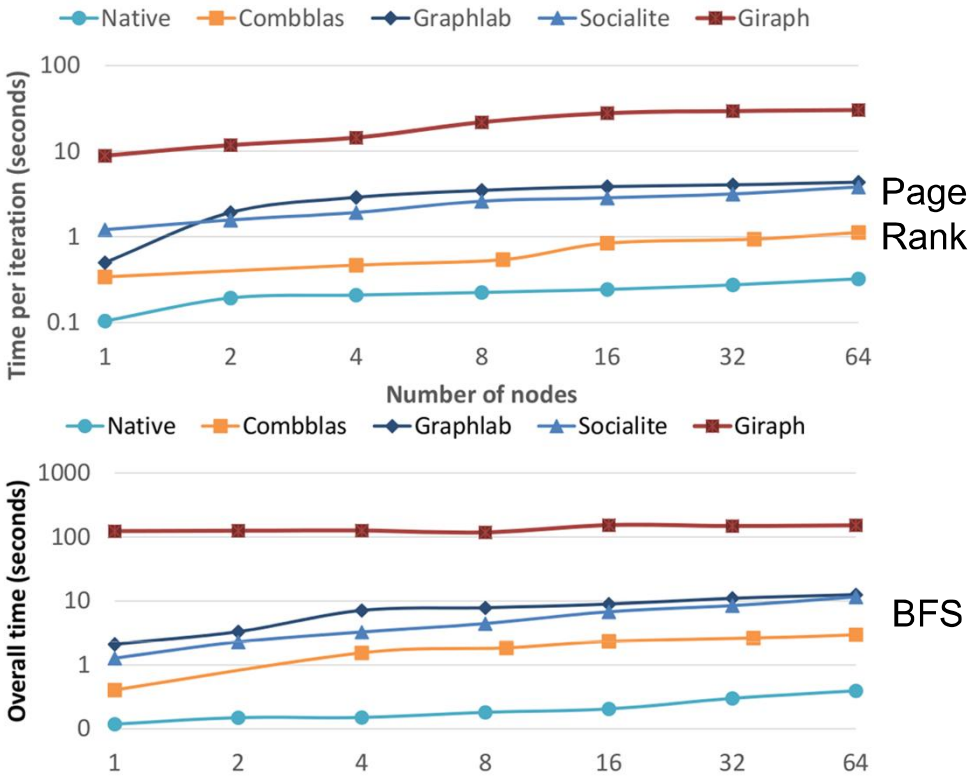
Performance (single node)

Algorithm	CombBLAS	GraphLab	SociaLite	Giraph	Galois
PageRank	1.9	3.6	2.0	39.0	1.2
BFS	2.5	9.3	7.3	567.8	1.1
Coll. Filtering	3.5	5.1	5.8	54.4	1.1
Triangle Count.	33.9	3.2	4.7	484.3	2.5

- CombBlas, Graphlab and SociaLite perform well on average
- CombBlas ran out of memory on Triangle Count (A^2 calculations)



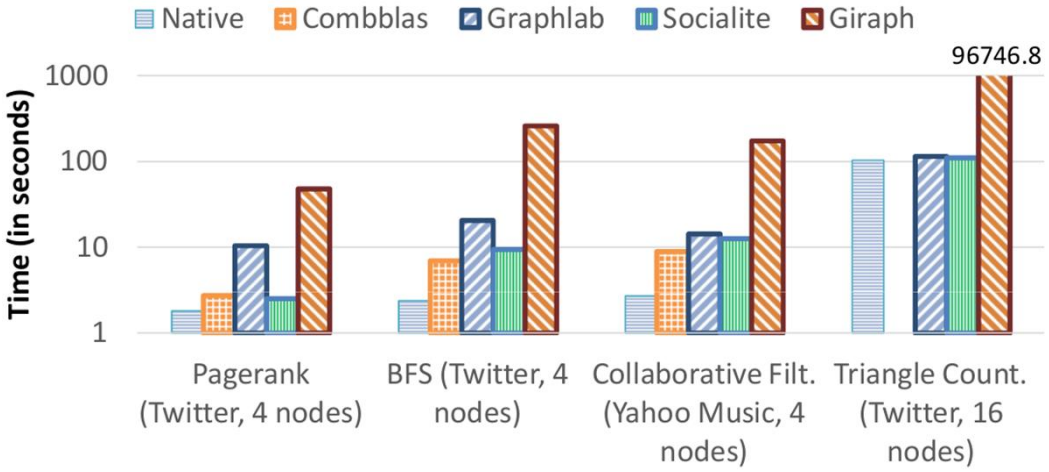
Performance (multiple nodes, synthetic)



- *"weak scaling"*: graph data per node constant,
- horizontal line denotes perfect scaling

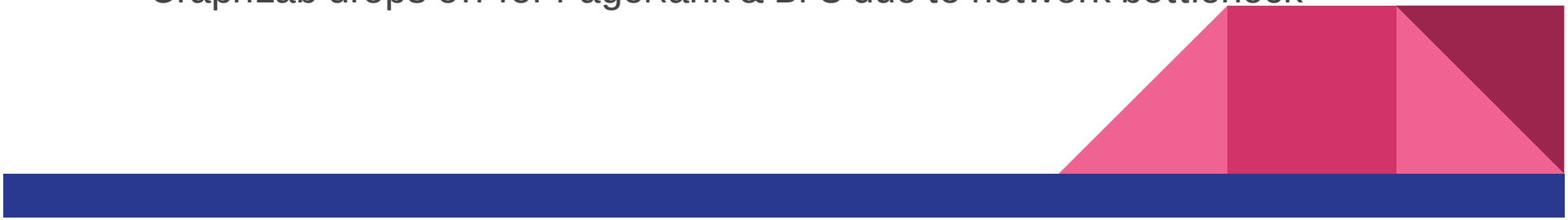
multiple node (real-world / combined)

Algorithm	CombBLAS	GraphLab	SociaLite	Giraph
PageRank	2.5	12.1	7.9	74.4
BFS	7.1	29.5	18.9	494.3
Coll. Filtering	3.5	7.1	7.0	87.9
Triangle Count.	13.1	3.6	1.5	54.4



Observations

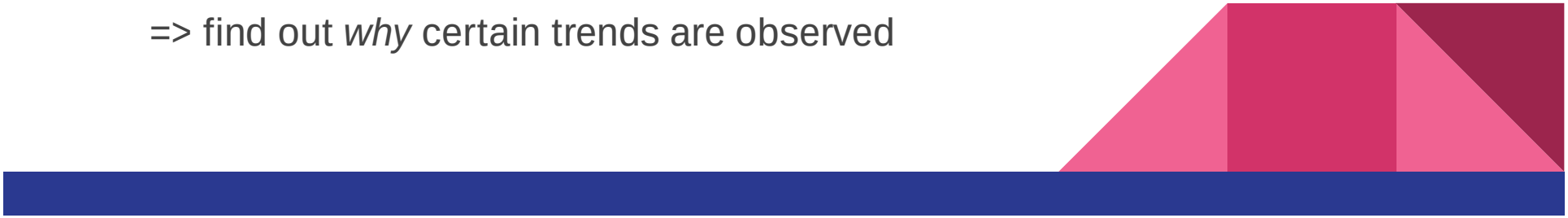
- Again: Native best, Giraph worst, CombBLAS OOM.
- Graphlab & SociaLite perform well on △ - counting because data structure is optimized for UNION - operations
- CombBLAS performs well for the other three algorithms
- GraphLab drops off for PageRank & BFS due to network bottleneck



Further Analysis: Resource monitoring

- CPU utilization (> larger is better)
- Peak network transfer rate (>)
- memory footprint (<)
- network data volume (<)

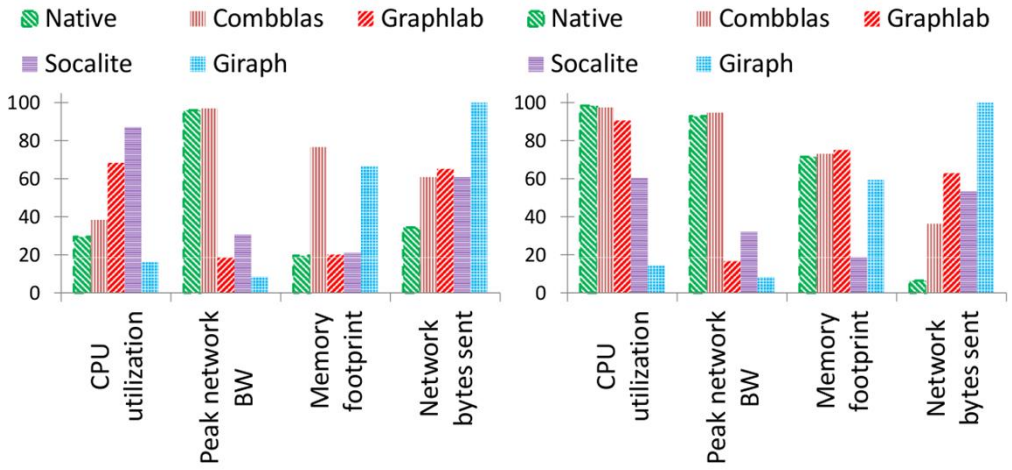
=> find out *why* certain trends are observed



Resource monitoring

- Giraph - only 16 % CPU due to memory requirements for each worker
- Pagerank: limited by *network traffic* for all ~> performance difference

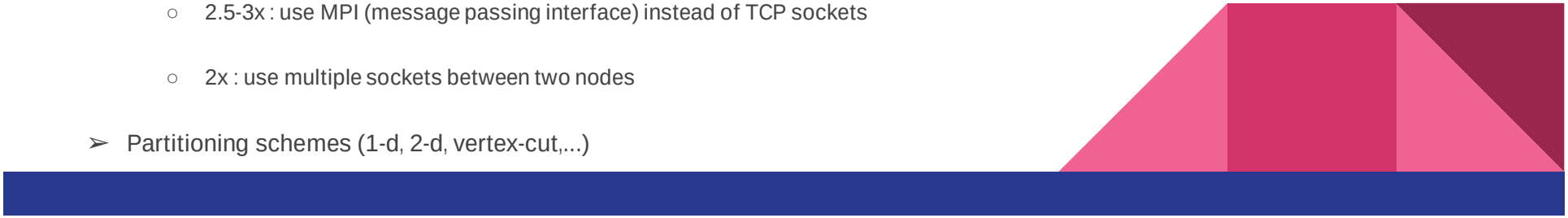
Algorithm	CombBLAS	GraphLab	SociaLite	Giraph
PageRank	2.5	12.1	7.9	74.4



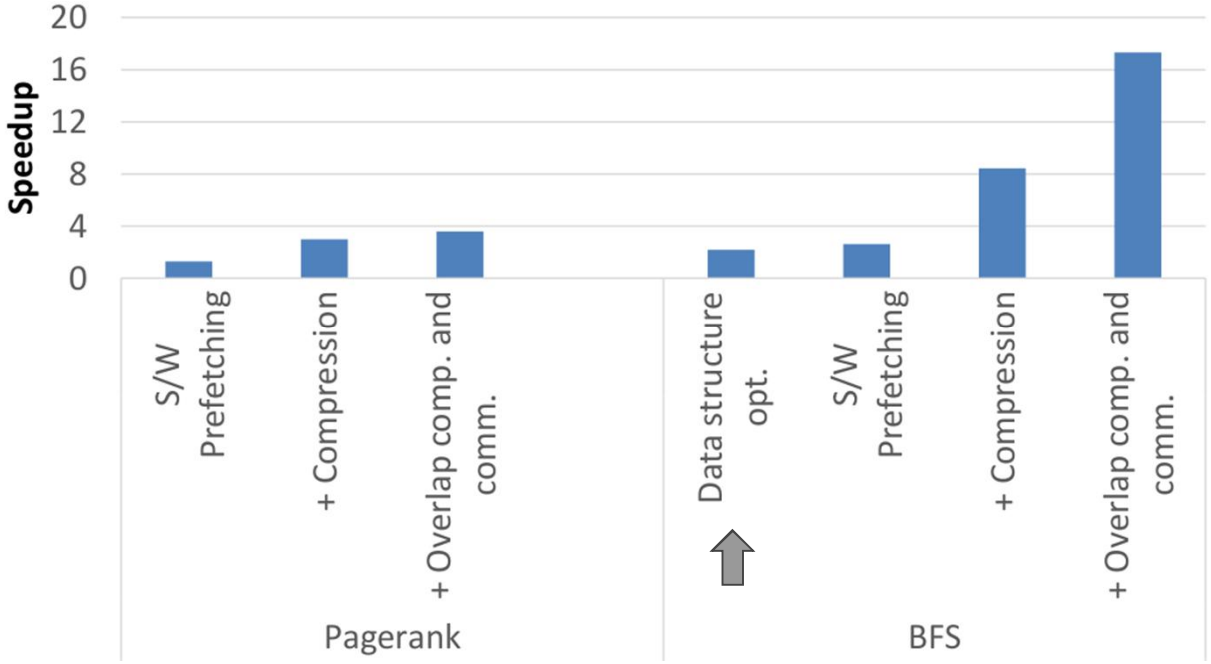
(a) PageRank (b) Breadth-First Search

Optimization

- Data Structure
  - 2x : bit vectors (exploit bit-level parallelism in hardware)
  - compressed sparse-row (CSR) format for adjacency list
- Data Compression
  - 2x : bit vectors + delta coding (store differences rather than actual values)
- Overlap of Computation and Communication
  - 1.2-2x : start computation before receiving whole message, chunk large messages
- Message passing mechanisms
  - 2.5-3x : use MPI (message passing interface) instead of TCP sockets
  - 2x : use multiple sockets between two nodes
- Partitioning schemes (1-d, 2-d, vertex-cut,...)



Optimization - PageRank & DFS



- optimization performed for the native algorithm

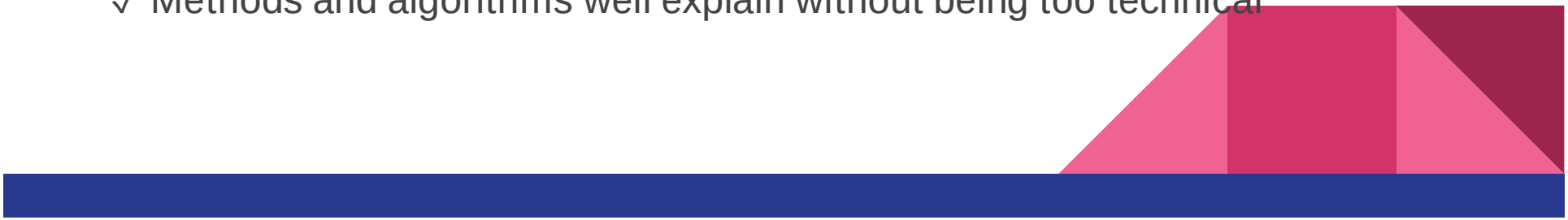
- DFS: bit-vectors for list of already visited vertices

Criticism & Summary

Criticism

- ➢ Not including spark-based GraphX (7x slower than GraphLab on PageRank)
- ➢ For those frameworks without pre-implemented code, your implementation might be too good/too bad and distort the results.

- ✓ Creates value for end-users and framework developers alike
- ✓ Methods and algorithms well explain without being too technical



Summary

- ➢ vertex-based: every vertex is an instance, communicates non-iterative
- ➢ Galois: best single node results. Giraph: slow
- ➢ Optimization: bit vectors, delta coding, CSR, Overlap, MPI
- ⇒ Reduce the performance gap through recommended changes and let the end user choose by preference.

Sources

[1] Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications? (Satish et al. ; 2012)

Images: Giraph, Graphlab, SociaLite, Galois

Introductory Paper: Graph Data Management Systems for New Application Domains (Cudré-Mauroux, Elnikety; 2011)

Main Paper, other tables and figures: Navigating the maze of graph analytics frameworks using massive graph datasets (Satish et al.; SIGMOD 2014)