Introduction

- In the past years several graph processing systems emerged. Graphs are growing fast and are becoming increasingly popular. Many problems can be modeled and solved using graphs.
- Comparison of non-uniform memory access (NUMA) aware systems and Giraph in their performance
  - on different graphs (real world and synthetic)
  - and different algorithms (SSSP, BFS, PR)

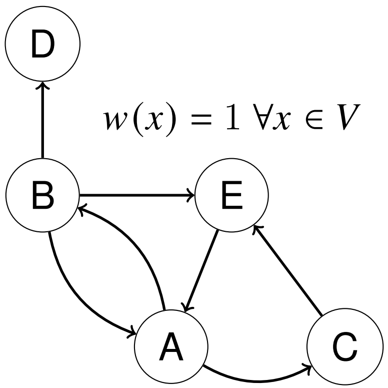
Overview

Preliminaries

A *weighted, directed graph* is the tuple $G = (V, E, w)$ where the *vertex set* is $V \subseteq \mathbb{N}$ and the E is the *edge set* with

$$E \subseteq \{(x, y) \mid x, y \in V, x \neq y\}$$

and $w : E \to \mathbb{R}$ is a mapping of edge to a weight.



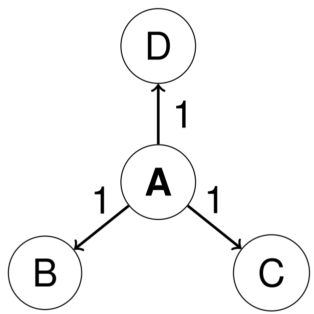
Algorithms

Single-Source Shortest-Paths (SSSP): find the shortest path from a starting vertex to every other vertex

Breadth-first search (BFS): find a node outgoing from a starting vertex, by increasing maximum hop count step-wise

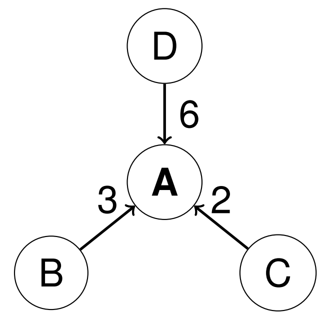
PageRank (PR): link analysis algorithm; weighs vertices, measuring their relative importance

Push Style



- reads active vertex, writes neighborhood
- more efficient, if only few active vertices at the same time
- more efficient, if neighborhoods do not overlap

Pull Style



- reads neighborhood, writes active vertex

→ only one write and many read operations

- less synchronization in parallel implementations needed
- more efficient, if many vertices active at the same time

Hugepages

- most systems use virtual memory management
  - represents an abstraction to hardware memory
  - virtual memory is then organized in pages
  - translations of virtual memory to physical memory are cached, because every translation takes time
- typically, memory pages are 4 KiB in size
- **hugepages** can be several MiB in size → reduce number of cache misses
- especially noticeable in very memory intensive applications

Frameworks

- **Galois** is a general purpose library designed for parallel programming
  - Version 6.0 from 29th June 2020 used
- **Gemini** uses a distributed message-based approach from scratch
  - Version from 2nd November 2016 used
  - Version contains bugs that had to be fixed
- **Giraph** is built on Apache Hadoop, a large scale data processing infrastructure
  - Version 1.3 from 8th May 2020 used
  - BFS is not natively supported
- **Ligra** dynamically switches between push and pull style
  - Version from 14th August 2019
- **Polymer** optimizes data layout and memory access strategies
  - Version from 28th August 2018

Evaluation

5 Machines, with

- 96 cores, of which 48 virtual
- 256 GB of RAM each, one machine only 128 GB
- Ubuntu 18.04.2 LTS

Measurements:

- **execution time**: time from start to finish of the console command
- **calculation time**: time the framework actually executed the algorithm
- **overhead**: time difference between execution time and calculation time (time to read the input graph, initialization, etc.)

Graph	# Vertices (M)	# Edges (M)
flickr	0.1	2
orkut	3	117
wikipedia	12	378
twitter	52	1963
rMat27	63	2147
friendster	68	2586
rMat28	121	4294

each test case (graph, framework, algorithm) was run 10 times

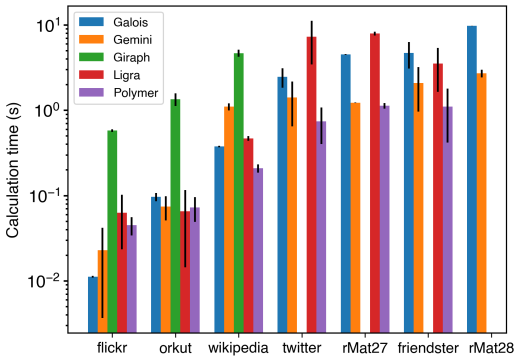
Production Case

- running system, that performs multiple calculations on a single graph
- without the need of reloading graph data with every calculation
- short calculation times should be preferred because the overhead time is only spent once on startup and amortizes quickly

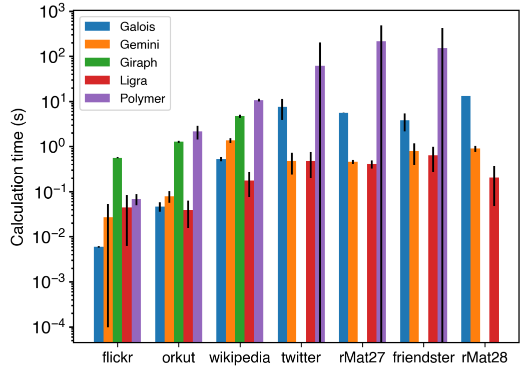
Research Case

- individual calculations on a graph, i.e. for each calculation, the graph has to be loaded
- the algorithm can change frequently
- requiring the framework to be relatively fast on different algorithms
- overall small execution times and small overhead are preferred

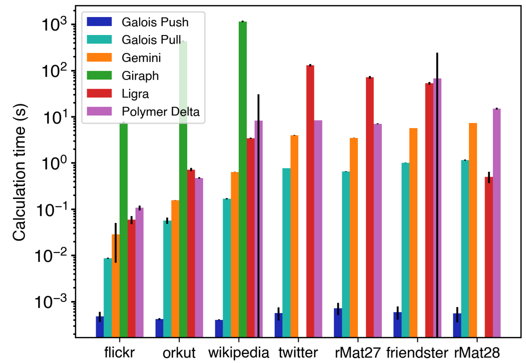
Production Case Single Node



(a) SSSP



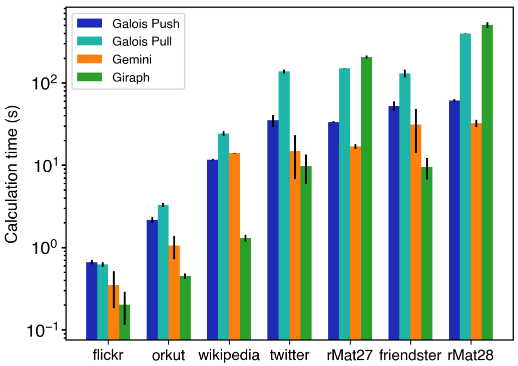
(b) BFS



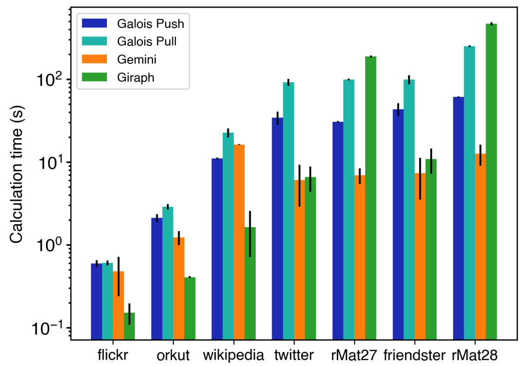
(c) PR

- Giraph is either very slow or requires too much RAM (>256 GB)
- On SSSP, Polymer is fastest, followed by Gemini on second place
- On BFS, Gemini and Ligra are comparable and fastest on the larger graphs
- On PR, Galois is fastest. But we exclude Galois Push because of possible measuring errors.
- Message-based approach can compete with shared-memory

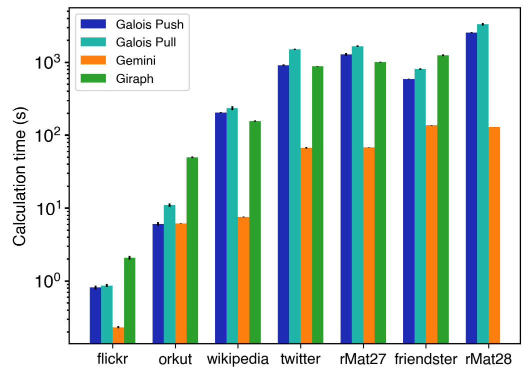
Production Case Distributed



(a) SSSP



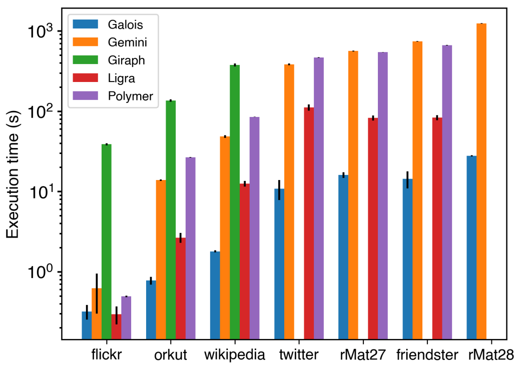
(b) BFS



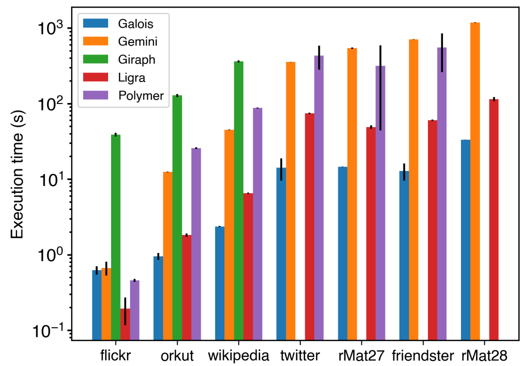
(c) PR

- Giraph is fastest on SSSP and BFS on the real world graphs
- Giraph has problems with synthetic graphs
- Gemini is fastest on PR, with Giraph on second place

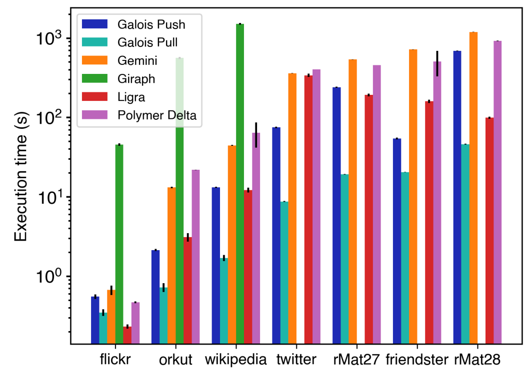
Research Case Single Node



(a) SSSP



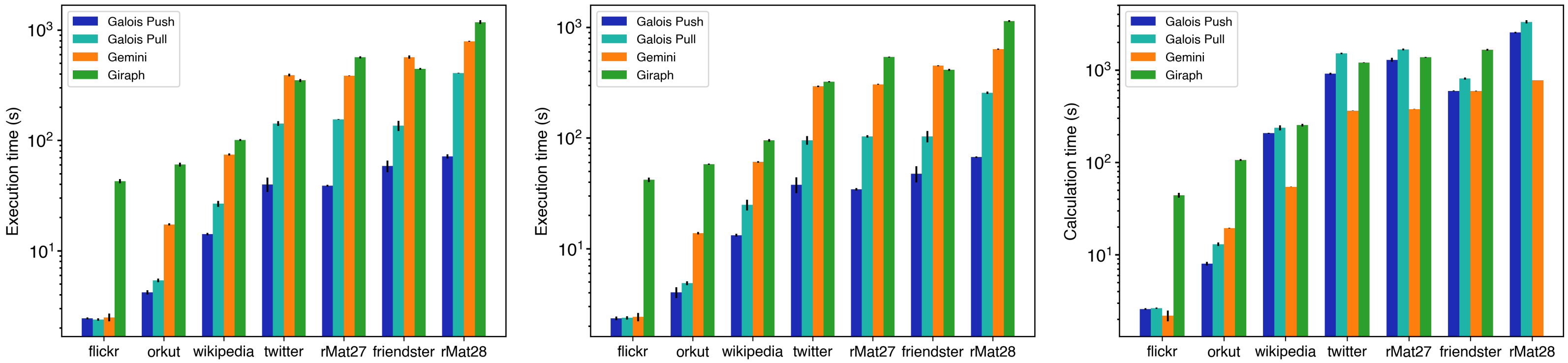
(b) BFS



(c) PR

- Giraph is either slowest or requires too much RAM (>256 GB)
- Galois is fastest in almost all cases, second fastest is Ligra
- Gemini and Polymer are comparably slow

Research Case Distributed

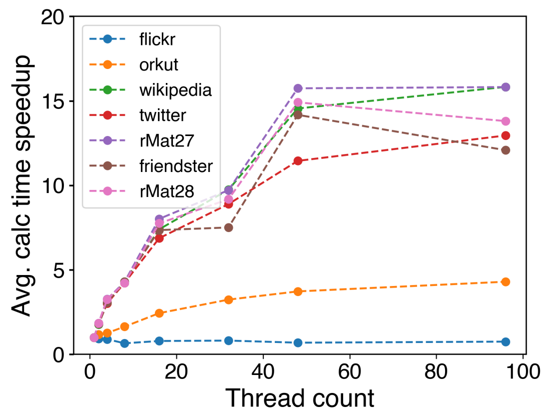


(a) SSSP (b) BFS (c) PR

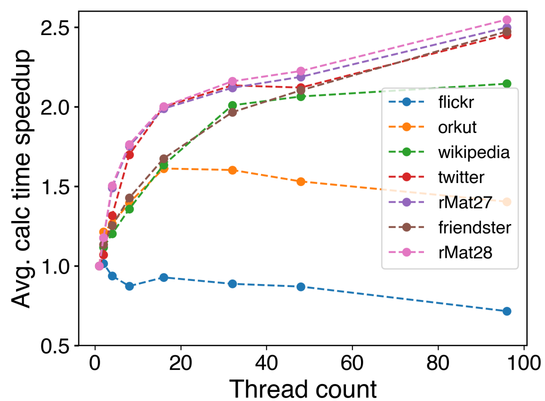
- Galois Push is faster than Pull in all cases
- Either Galois implementation is faster than any other frameworks on SSSP or BFS
- Gemini is fastest on PR and comparable to Giraph on SSSP and BFS

Galois With Hugepages

	Graph	Calc Time (s) w/o	Calc Time (s) w/	Exec Time (s) w/o	Exec Time (s) w/
SSSP	flickr	0.01	**0.01**	0.3	**0.2**
	orkut	0.10	**0.02**	0.8	**0.5**
	wikipedia	0.38	**0.11**	1.8	**1.1**
	twitter	2.47	**0.94**	10.8	**5.1**
	rMat27	4.50	**1.39**	16.0	**6.4**
	friendster	4.70	**1.78**	14.4	**7.5**
	rMat28	9.77	**3.34**	27.8	**13.1**
PR Pull	flickr	0.01	**0.01**	0.3	**0.2**
	orkut	0.06	**0.02**	0.7	**0.6**
	wikipedia	0.17	**0.03**	1.7	**1.4**
	twitter	0.77	**0.11**	**8.7**	9.3
	rMat27	0.65	**0.13**	19.2	**8.1**
	friendster	1.01	**0.14**	20.4	**13.1**
	rMat28	1.15	**0.24**	46.0	**16.4**



SSSP



PR Pull

Conclusion and Outlook

- performance highly dependent on the framework, algorithm and data set
  - Galois is almost always fastest in the research case; especially with hugepages
  - Giraph is good on SSSP or BFS in distributed production
  - Gemini is a good middleground for distributed PR and single node production
- single node almost always preferrable, as long as RAM is sufficient

Outlook

→ incorporate new frameworks and new algorithms

→ great range of settings and multiple implementations for the same problem

→ At a later point in time, it is important to repeat such a comparison, because the frameworks are further developed and new ones are created.