Comparison of Graph Processing Systems

Tuesday, 20th October 2020

Motivation and Goal

- graph processing becomes increasingly important in academic and industrial environments
- many problems modeled with graphs, e. g., machine learning and data mining
- many business models are based on graphs, e. g., viral marketing or Google's search engine
- graph sizes increase to several billion edges

→ performance, parallelism and distribution of graph algorithms becomes more important

Main Goal: *Comparison of five graph processing systems in their performance on different graphs and algorithms.*

Overview

1. Preliminaries
    - Basics
    - Computation Styles
    - Hugepages
2. Frameworks
3. Evaluation
    - Research vs. Production Case
    - Results
4. Conclusion and Outlook

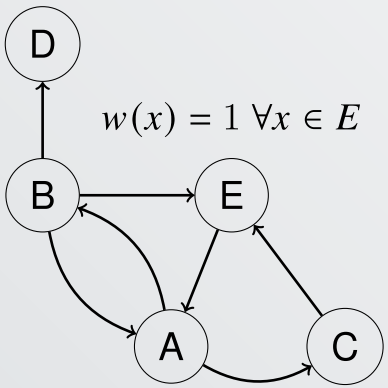
Preliminaries

Graphs

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.



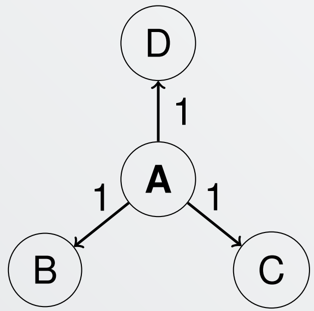
Algorithms

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

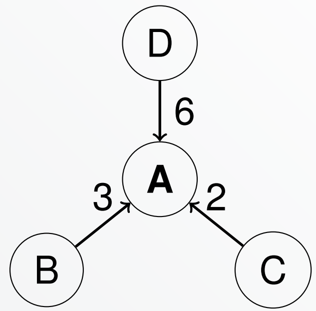
PageRank (PR): link analysis algorithm; weighs vertices, measuring their relative importance

Push Style



- reads active vertex, writes neighborhood
- more efficient, if only few active vertices at the same time
- more efficient, if neighborhoods of active vertices 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 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 $\rightarrow$ reduce number of cache misses
- especially noticeable in very memory intensive applications

Frameworks

Framework	Version	NUMA	Dist.	Features	Notes
Galois	29.06.2020	✓	(✓)	general purpose library designed for parallel programming, Hugepage support	distributed using Gluon
Gemini	02.11.2016	✓	✓	distributed message-based approach from scratch	version contains bugs that had to be fixed
Giraph	08.05.2020	X	✓	built on Apache Hadoop	BFS is not natively supported
Ligra	14.08.2019	✓	X	dynamically switches between push and pull style	
Polymer	28.08.2018	✓	X	optimizes data layout and memory access strategies	

Evaluation

Machines

vsflash1-5,

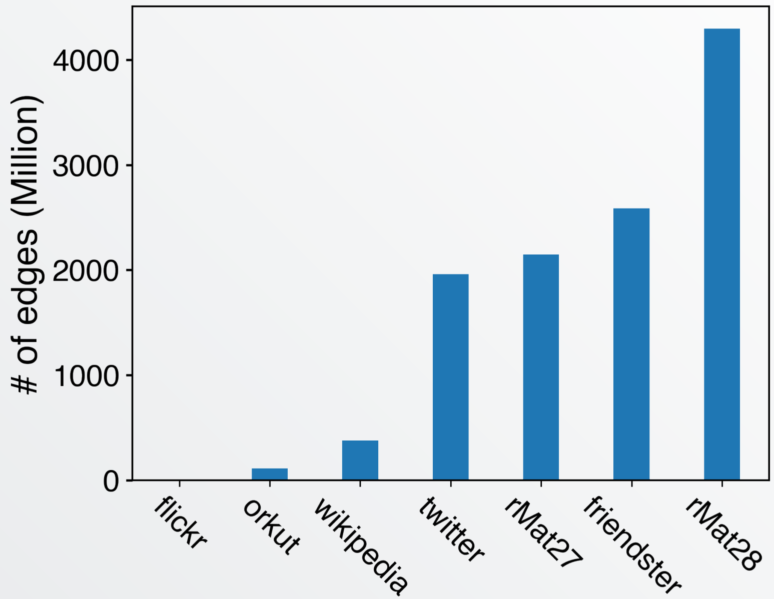
- 96 cores, of which 48 virtual
- 256 GB of RAM each[1]
- Ubuntu 18.04.2 LTS

Measurements

- **execution time**: time from start to finish of the console command
- **calculation time**: time the framework actually executed the algorithm
- executed each test case 10 times

Graphs

Both rMat graphs are synthetic, others are real-world data sets; Flickr: 24MB, rMat28: 76GB



[1] one machine only 128 GB

Production Case

- running system: multiple calculations on a single graph
- graph data stays loaded between calculations

→ short calculation times should be preferred

- Not main focus of this presentation![2]

Research Case

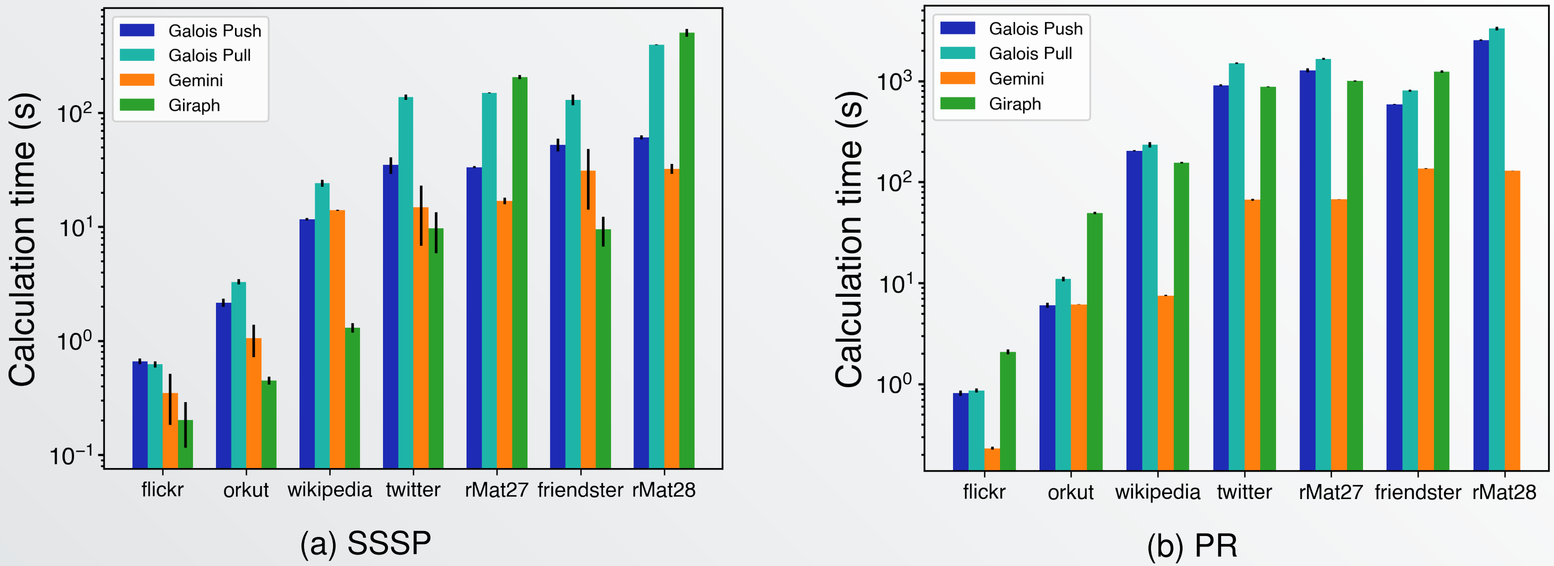
- individual calculation cases: possibly new graph for each calculation
- frequently changing algorithm

→ framework should be relatively fast on different algorithms

→ overall small execution times should be preferred

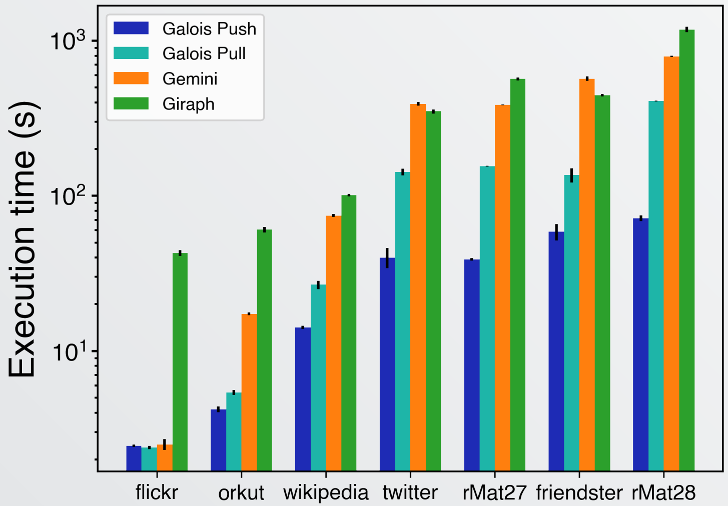
[2] see paper for details

Production Case Distributed

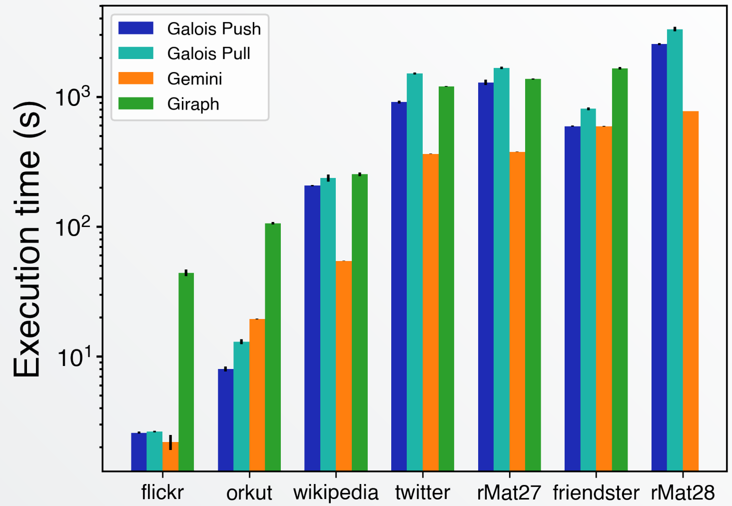


- Giraph is fastest on SSSP and BFS on the real world graphs
- Giraph has problems with synthetic graphs
- Gemini is fastest on PR, with Giraph on second place

Research Case Distributed



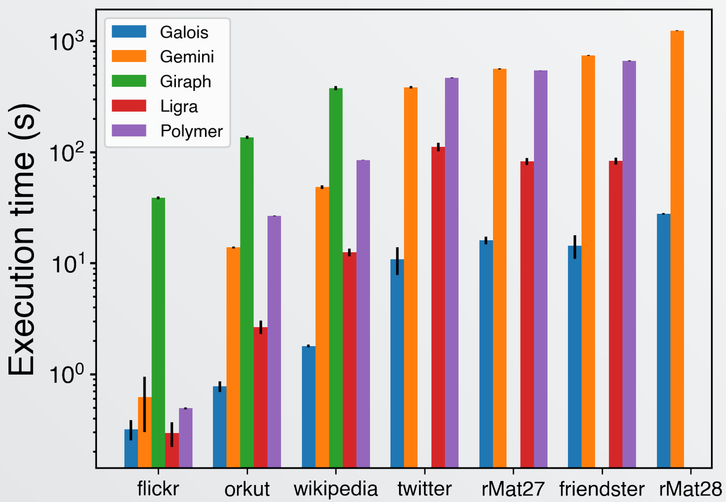
(a) SSSP



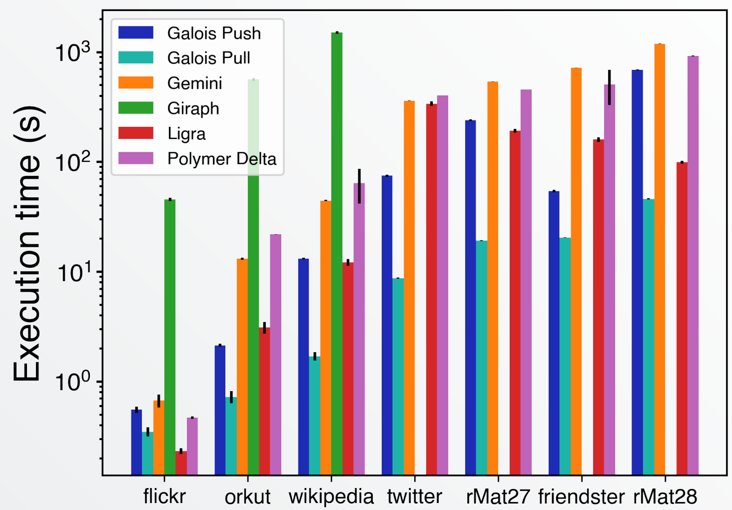
(b) PR

- Galois Push is faster than Pull in all cases
- Both Galois implementations fastest on SSSP or BFS
- Gemini is fastest on PR in almost all cases

Research Case Single Node



(a) SSSP



(b) 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

Galois With Hugepages

	Calc Time (s)		Exec Time (s)	
Graph	w/o	w/	w/o	w/
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**

(a) SSSP

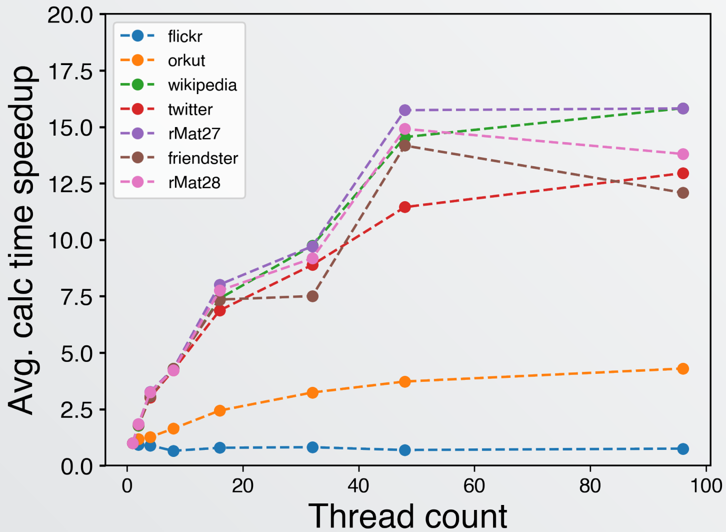
	Calc Time (s)		Exec Time (s)	
Graph	w/o	w/	w/o	w/
flickr	0.01	**0.01**	0.3	**0.2**
orkut	0.06	**0.02**	0.7	**0.6**
wikipedia	0.17	**0.03**	1.7	**1.4**
twitter	0.77	**0.11**	**8.7**	9.3
rMat27	0.65	**0.13**	19.2	**8.1**
friendster	1.01	**0.14**	20.4	**13.1**
rMat28	1.15	**0.24**	46.0	**16.4**

(b) PR Pull

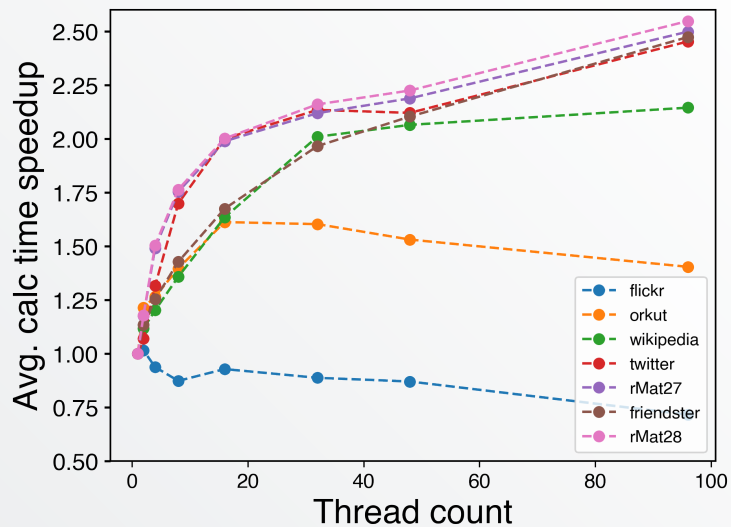
- Hugepages reduce both calculation and execution time on all algorithms

$\rightarrow$ Execution times can be up to 3$\times$ shorter

Multithreaded Speedup of Galois



(a) SSSP with Hugepages



(b) PR Pull with Hugepages

- Speedups can be significant, with and without hugepages
- Speedup of PR not to the same degree as on SSSP (2.5× vs. 15×)

Conclusion and Outlook

Generally: 1) performance highly dependent on the framework, algorithm and data set
2) single node almost always preferrable, as long as RAM is sufficient

Production Case

- Giraph is very fast on distributed systems (especially SSSP and BFS)
- Gemini is fast for distributed PR
- Gemini and Ligra are good options for single node

Research Case

- Galois is fastest in almost all cases; further improvements with hugepages possible

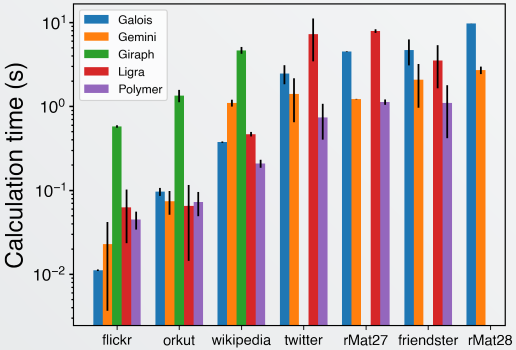
Outlook

→ incorporate new frameworks and new algorithms

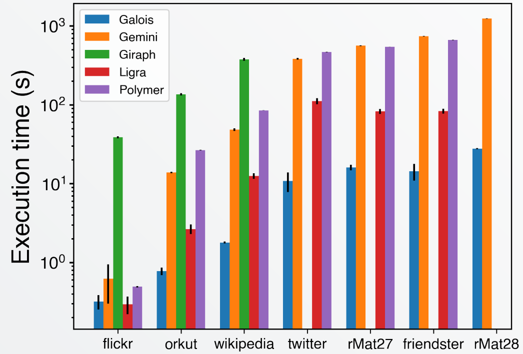
→ explore range of settings and other implementations

→ repeat similar tests in the future: frameworks are updated and new ones are introduced

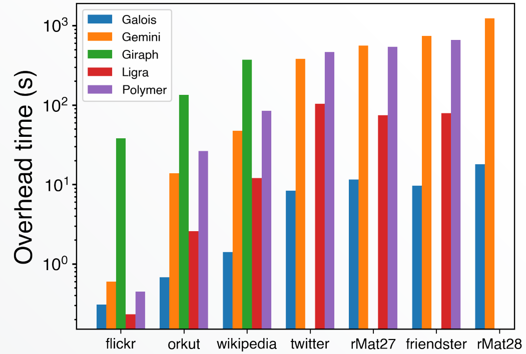
Additional Data



(a) Calculation times

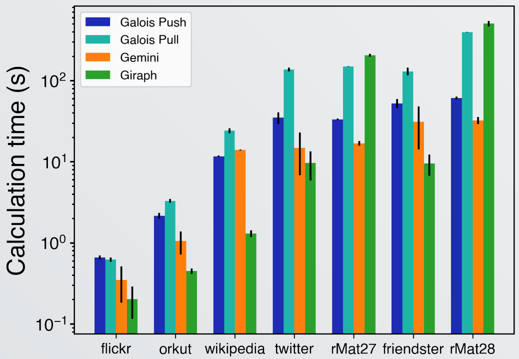


(b) Execution times

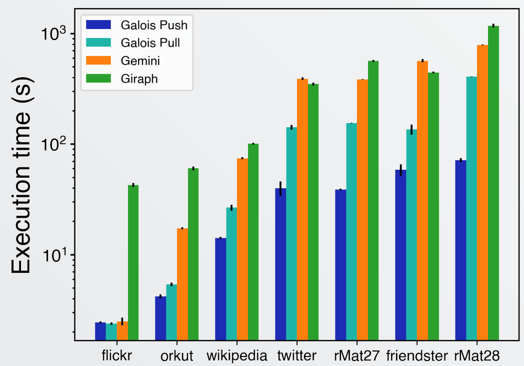


(c) Overhead time

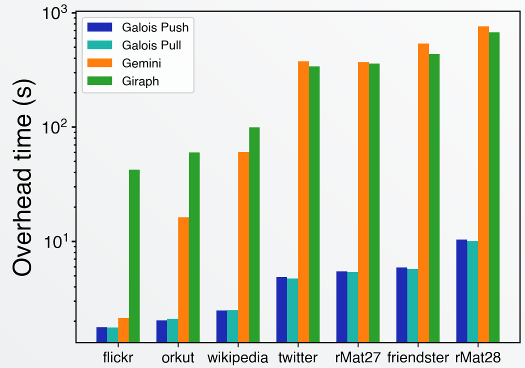
Figure 6: Average times for SSSP on a single computation node



(a) Calculation times

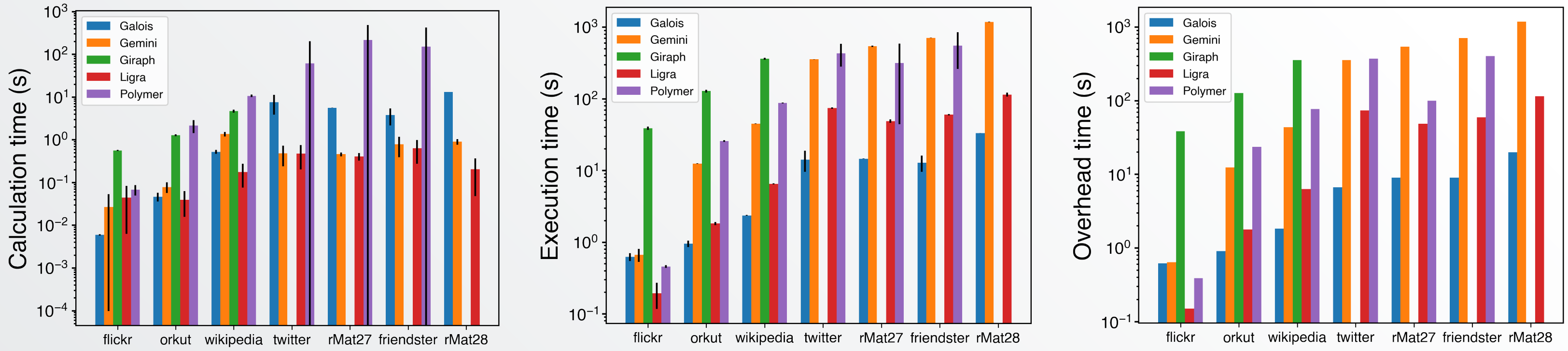


(b) Execution times



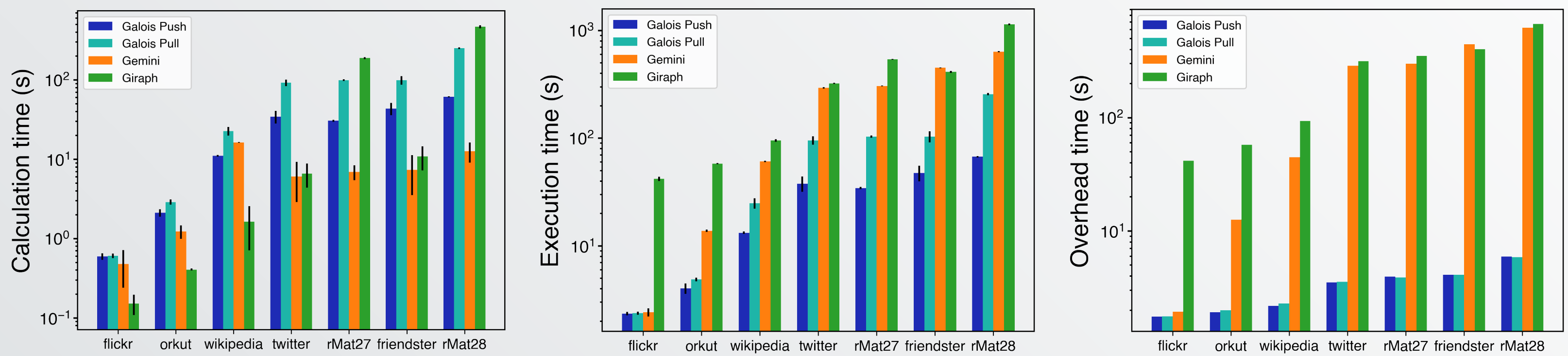
(c) Overhead times

Figure 7: Average times for SSSP on the distributed cluster



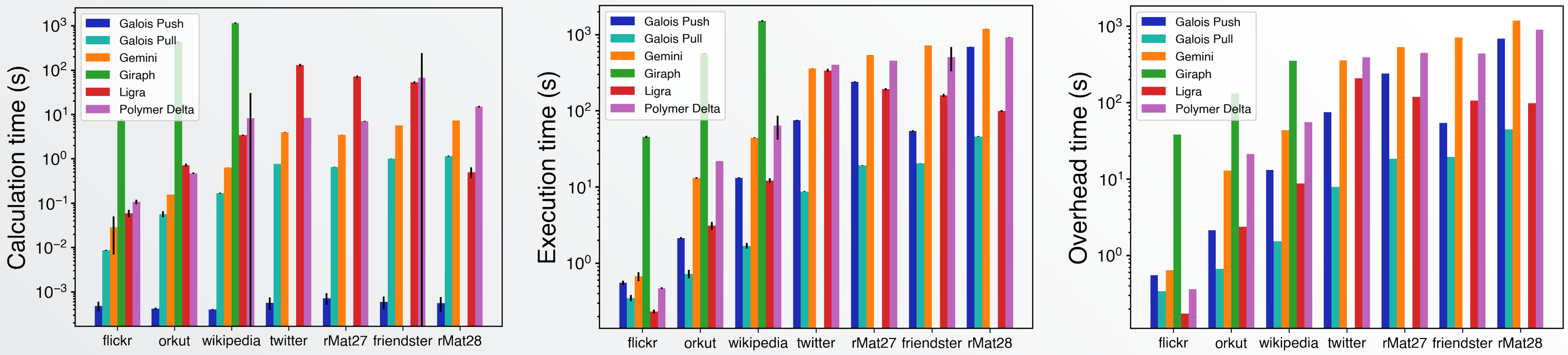
(a) Calculation time (b) Execution time (c) Overhead time

Figure 8: Average times for BFS on a single computation node



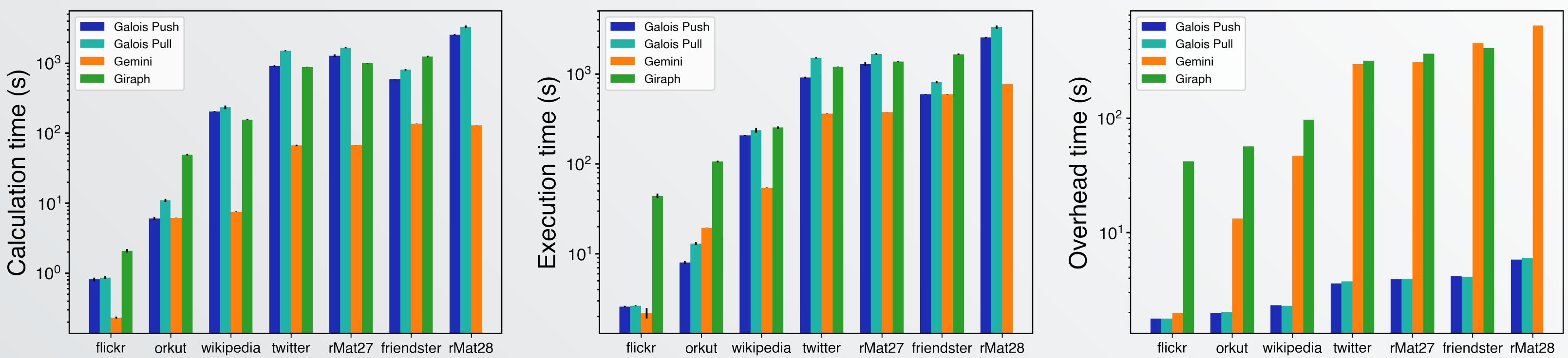
(a) Calculation time (b) Execution time (c) Overhead

Figure 9: Average times for BFS on the distributed cluster



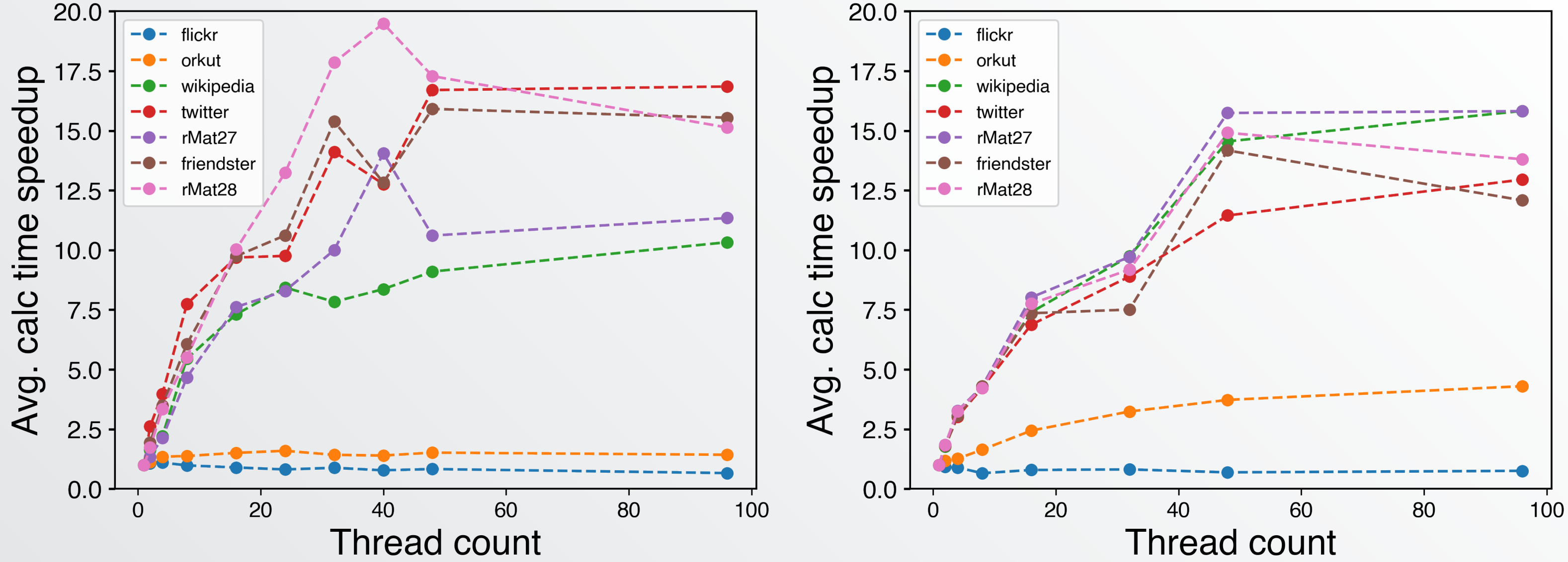
(a) Calculation time (b) Execution time (c) Overhead

Figure 10: Average times for PR on a single computation node



(a) Calculation time (b) Execution time (c) Overhead

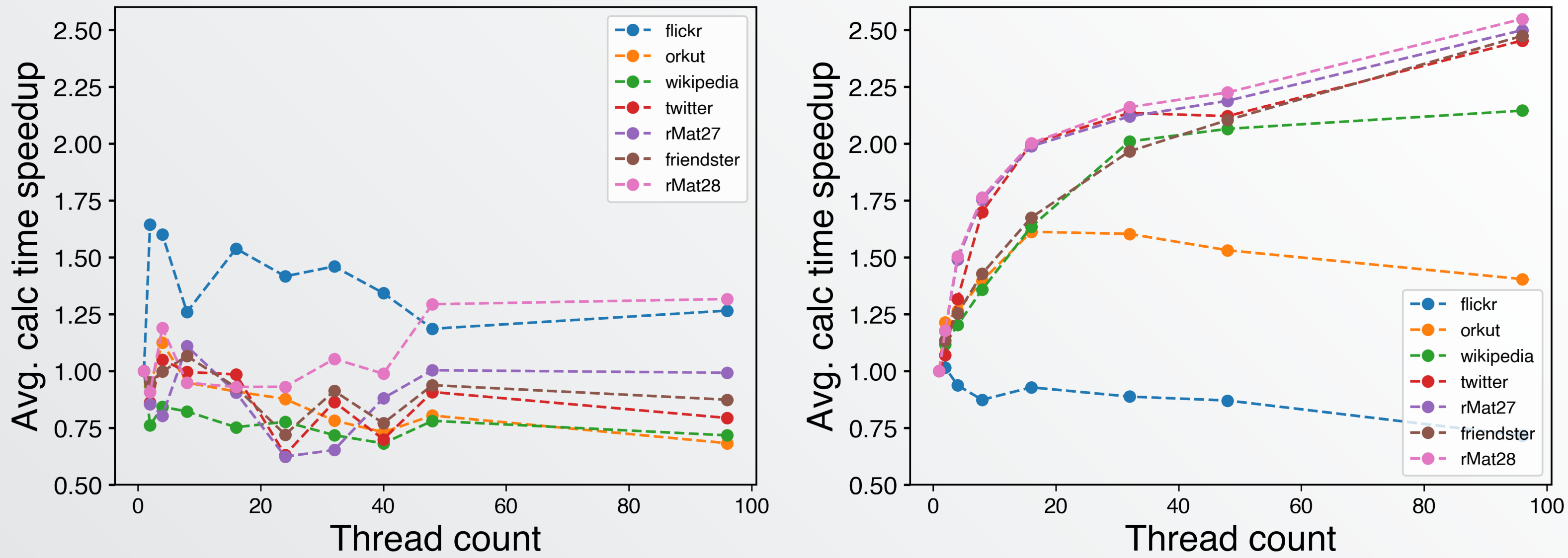
Figure 11: Average times for PR on the distributed cluster



(a) without hugepages

(b) with hugepages

Figure 12: Calculation time speedups on SSSP



(a) without hugepages

(b) with hugepages

Figure 13: Calculation time speedups on PR Pull