An Experimental Comparison of Pregel-like Graph Processing Systems

Minyang Han, Khuzaima Daudjee, Khaled Ammar,
M. Tamer Özsu, Xingfang Wang, Tianqi Jin

University of Waterloo

VLDB 2014

Overview

Overview

Motivation

Many new graph processing systems...
...but existing studies lack scale and comprehensiveness.

Our study provides:

- Experiments with **scale**: up to 128 EC2 machines.
- **Comprehensive** combination of algorithms and datasets.
- Focus on **time**, **memory**, and **network**.
- Use of similar systems for an **apples-to-apples comparison**.

Motivation

Many new graph processing systems...
...but existing studies lack scale and comprehensiveness.

Our study provides:

- Experiments with **scale**: up to 128 EC2 machines.
- **Comprehensive** combination of algorithms and datasets.
- Focus on **time**, **memory**, and **network**.
- Use of similar systems for an **apples-to-apples comparison**.

Motivation

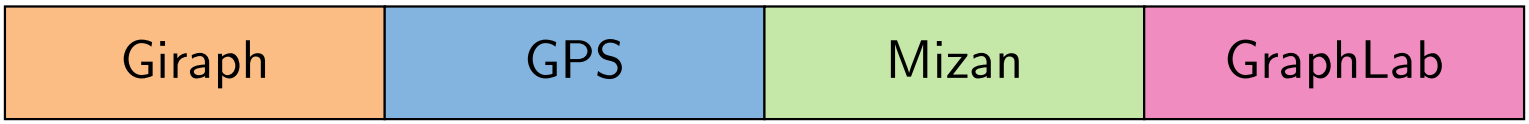
Many new graph processing systems...
...but existing studies lack scale and comprehensiveness.

Our study provides:

- Experiments with **scale**: up to 128 EC2 machines.
- **Comprehensive** combination of algorithms and datasets.
- Focus on **time**, **memory**, and **network**.
- Use of similar systems for an **apples-to-apples comparison**.

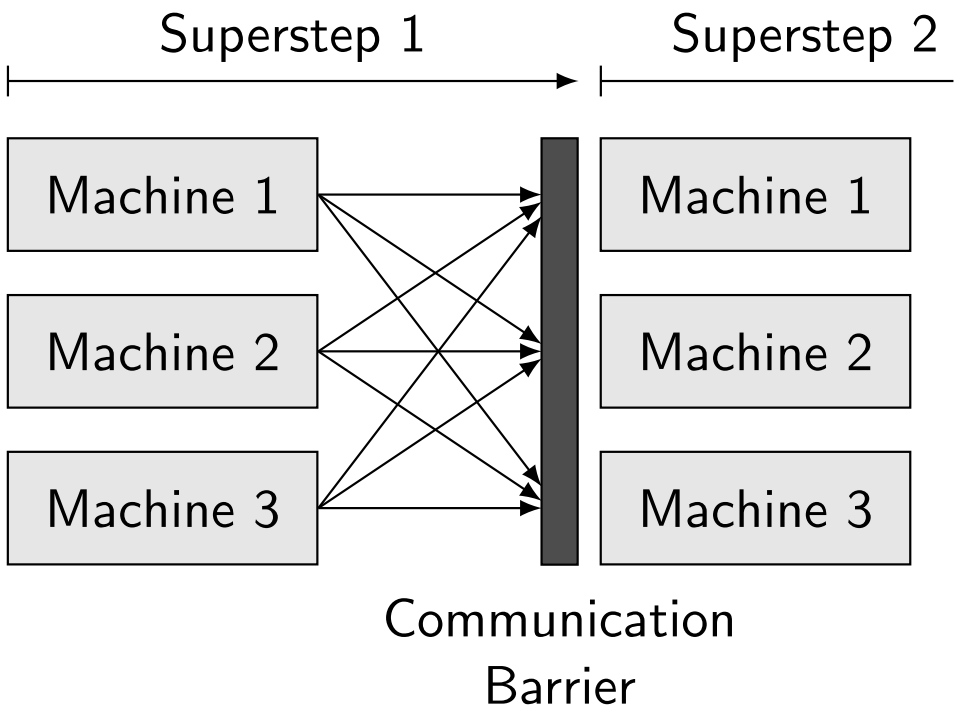
Overview

Systems Tested

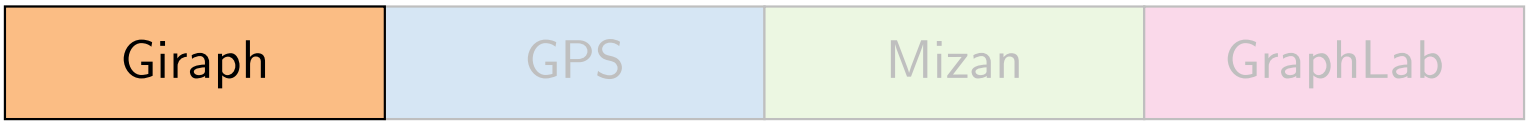


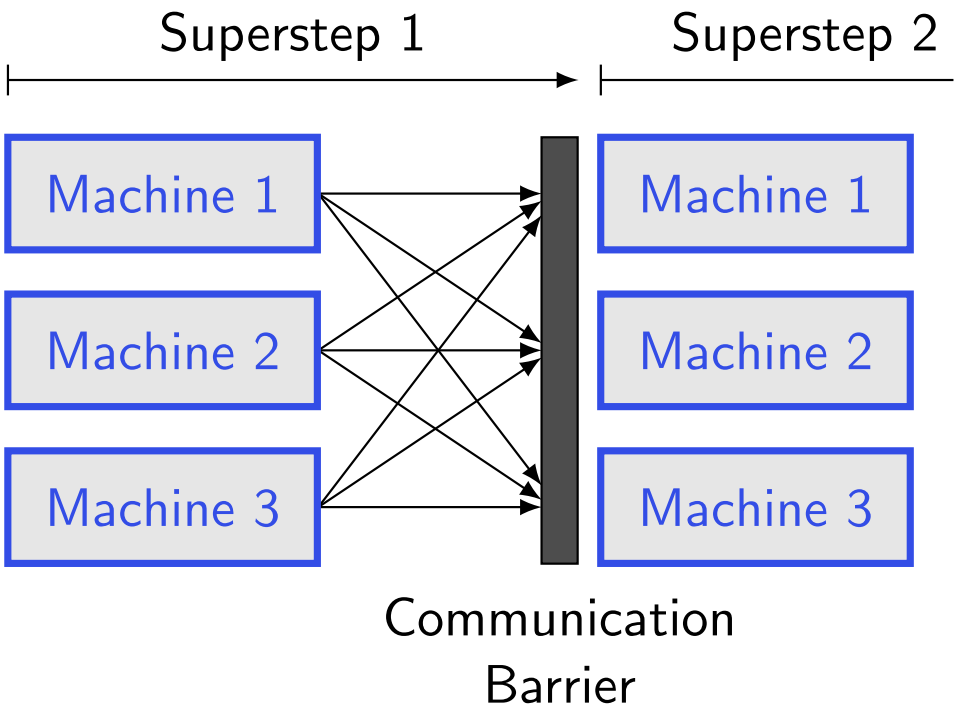
Systems Tested

Giraph	GPS	Mizan	GraphLab

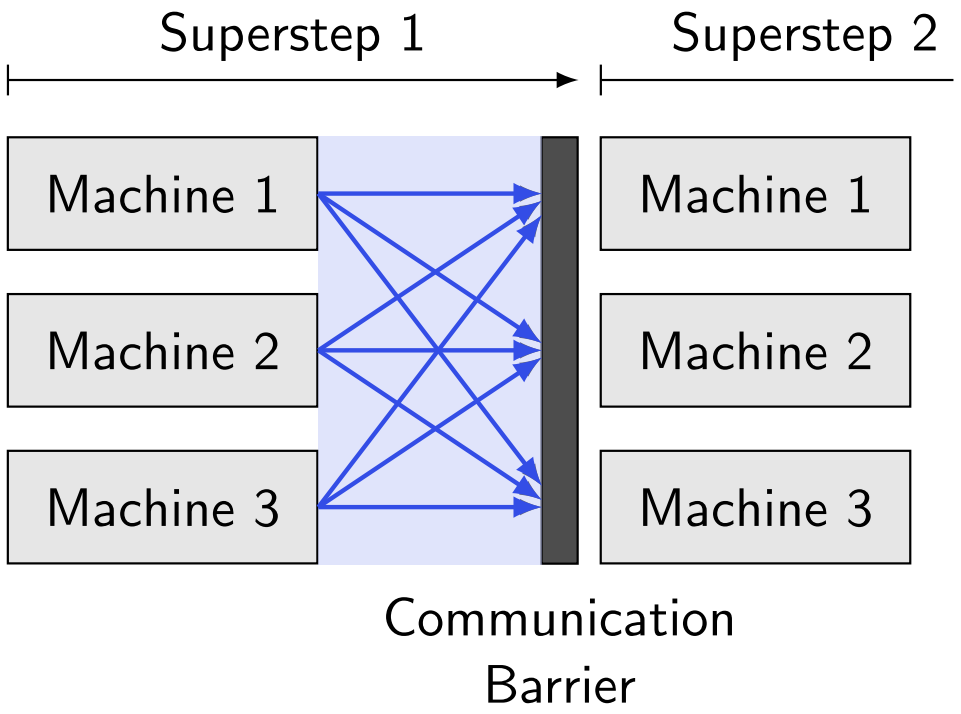


Systems Tested

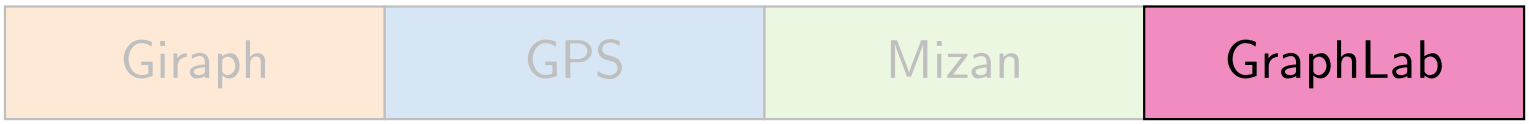


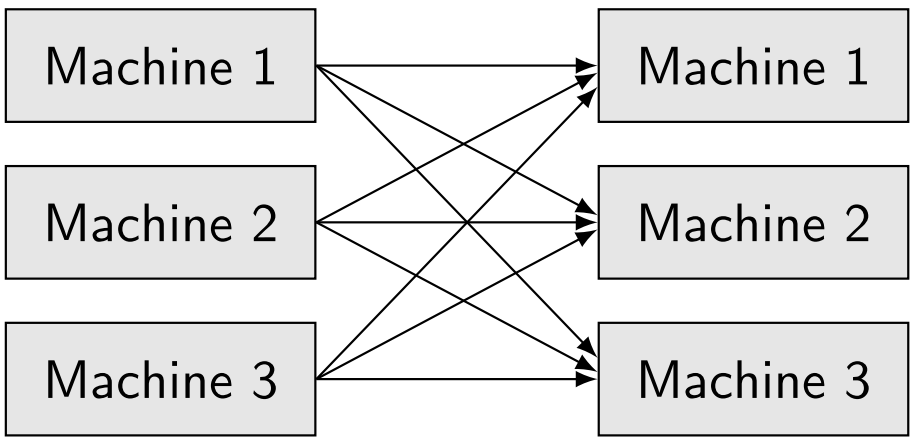


Systems Tested

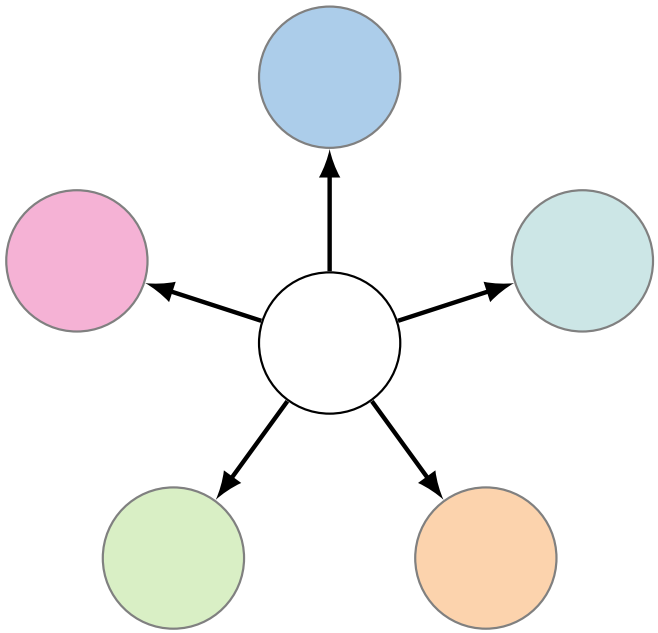


Systems Tested

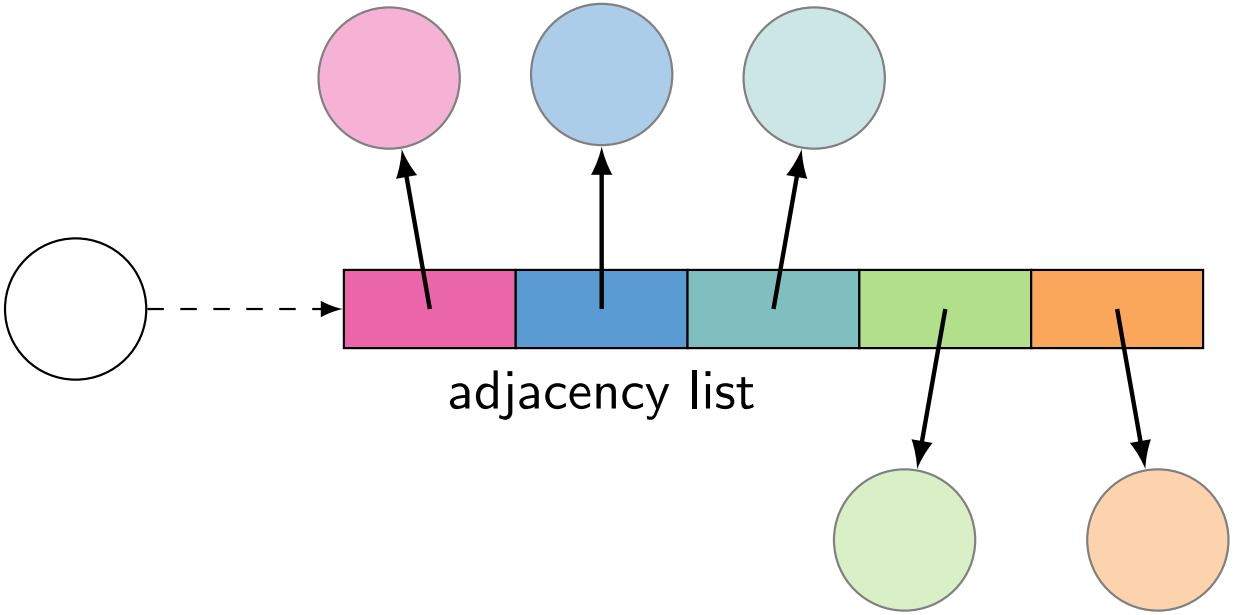




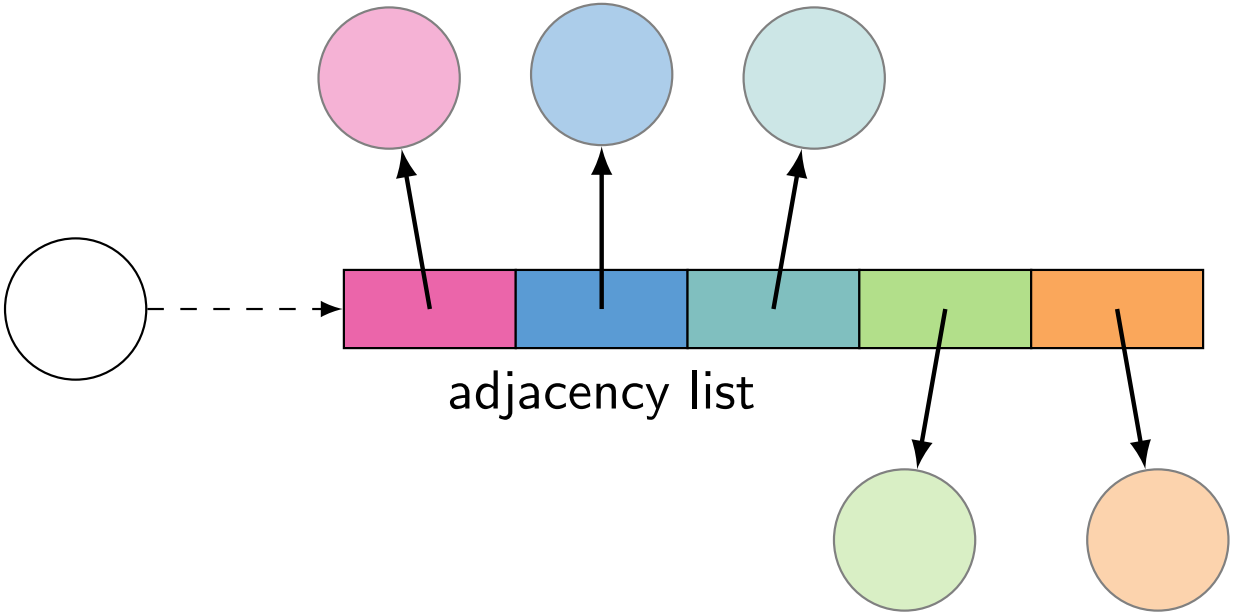
Giraph



Giraph

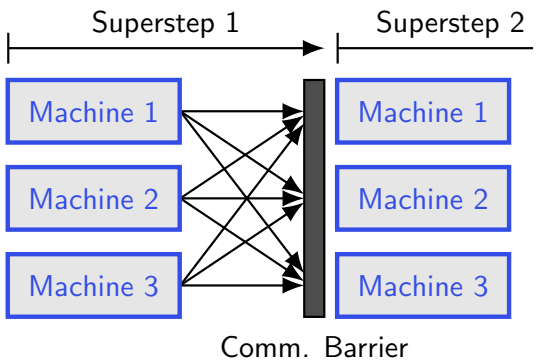


Giraph



Can be implemented as:

- Byte array (default).
- Hash map.

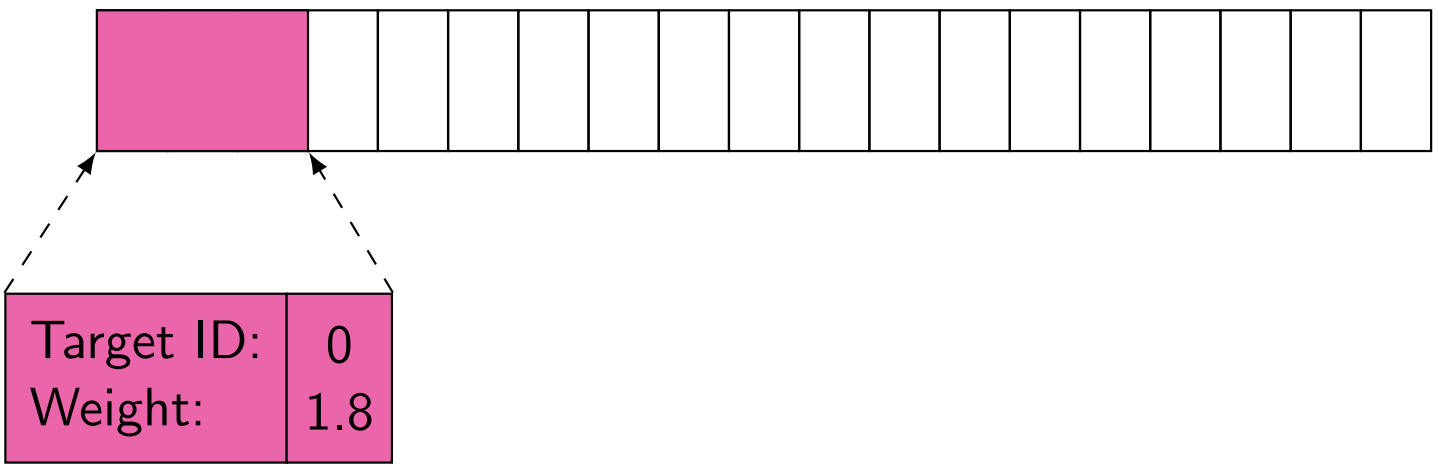


Giraph (Byte Array)

Byte array adjacency list:

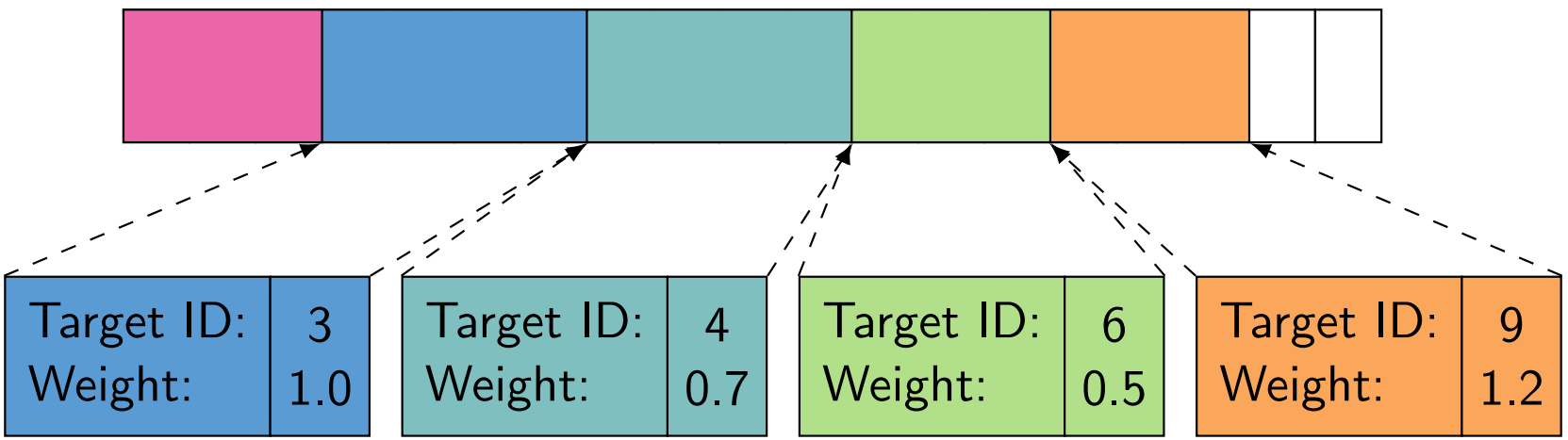
Giraph (Byte Array)

Byte array adjacency list:



Giraph (Byte Array)

Byte array adjacency list:



Giraph (Byte Array)

Byte array adjacency list:

Giraph (Byte Array)

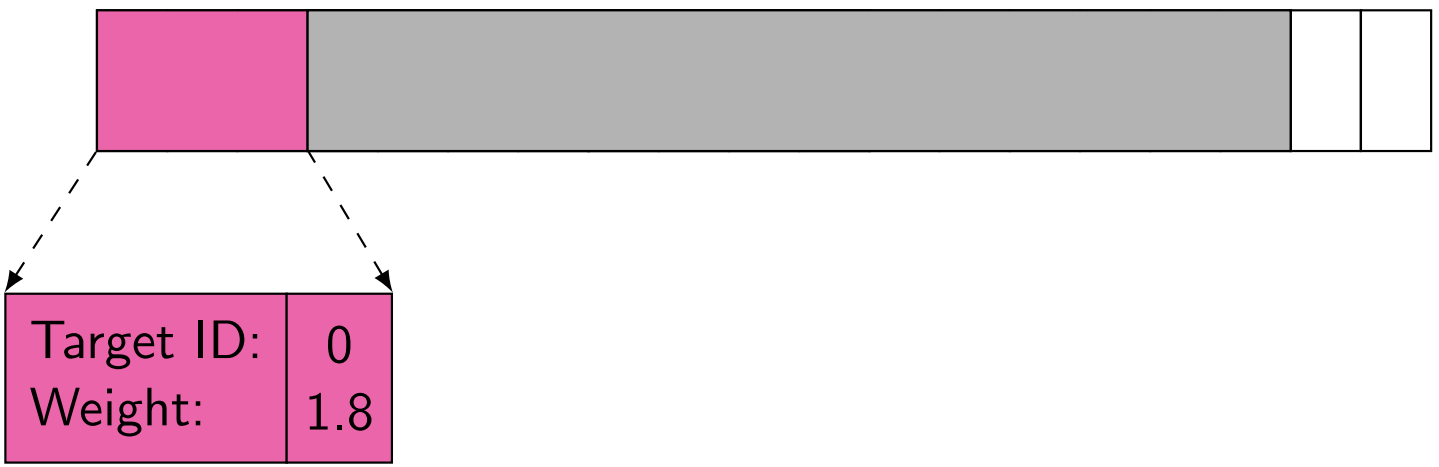
Byte array adjacency list:

Giraph (Byte Array)

Byte array adjacency list:

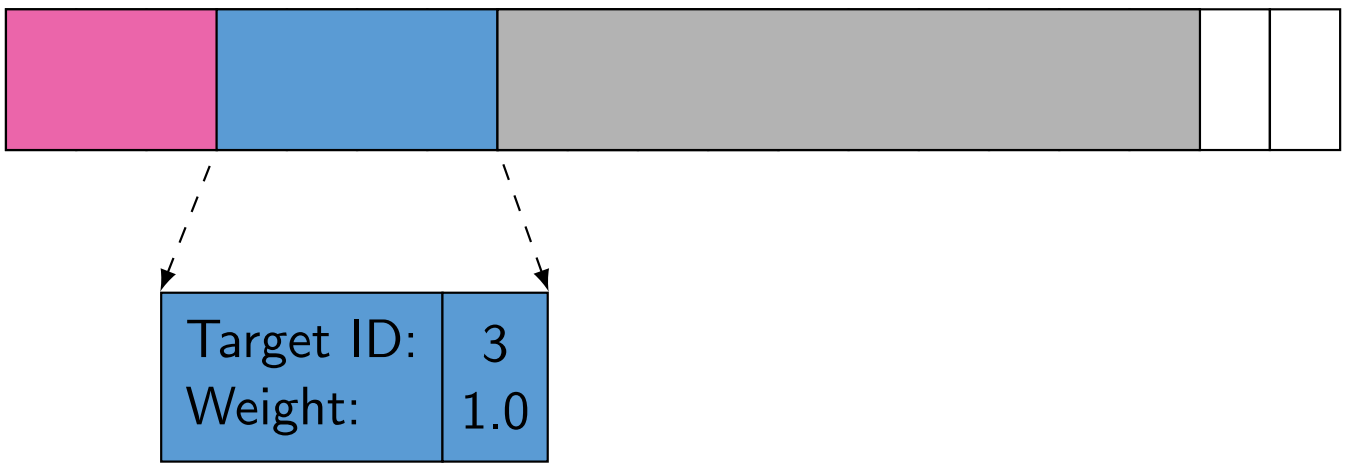
Giraph (Byte Array)

Byte array adjacency list:



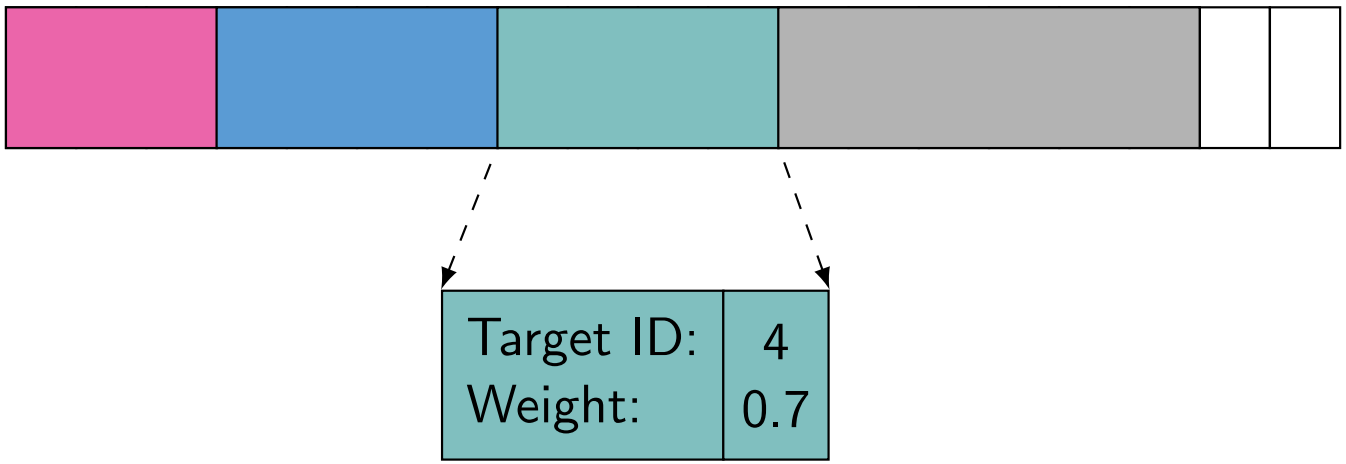
Giraph (Byte Array)

Byte array adjacency list:



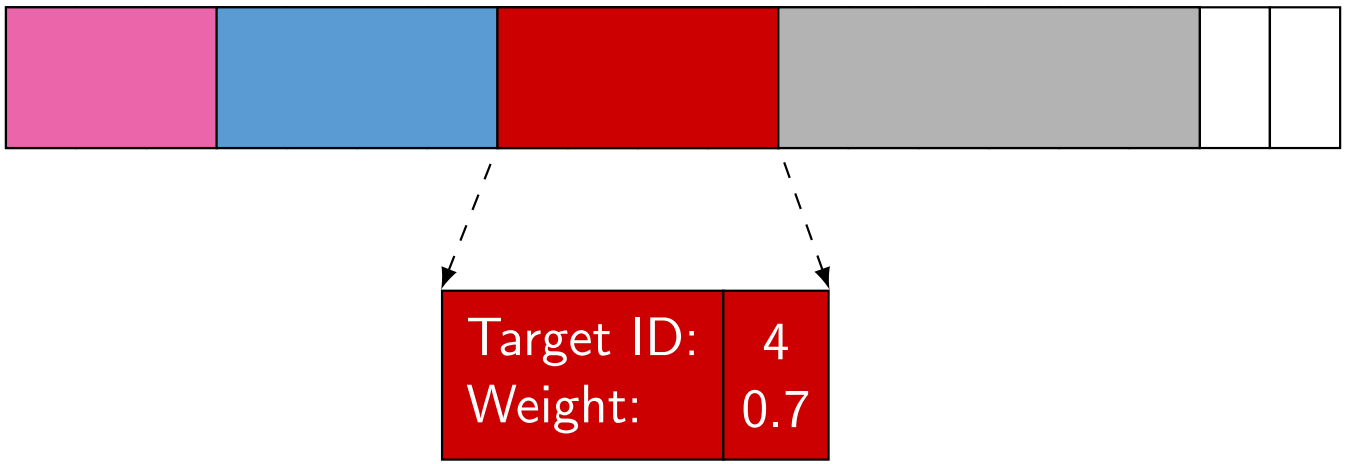
Giraph (Byte Array)

Byte array adjacency list:



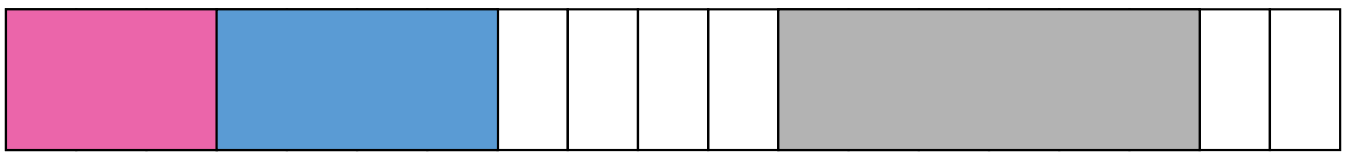
Giraph (Byte Array)

Byte array adjacency list:



Giraph (Byte Array)

Byte array adjacency list:

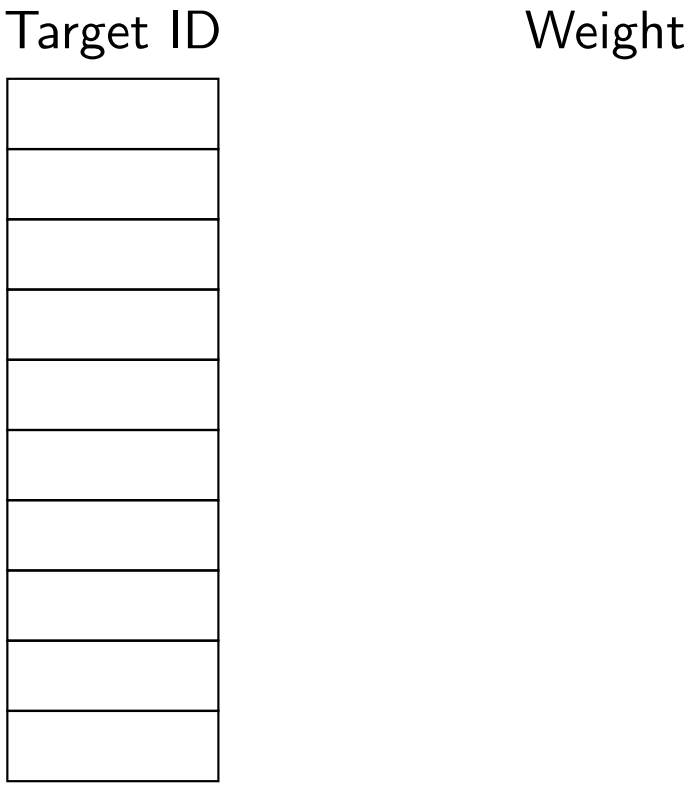


Giraph (Byte Array)

Byte array adjacency list:

Giraph (Hash Map)

Hash map adjacency list:



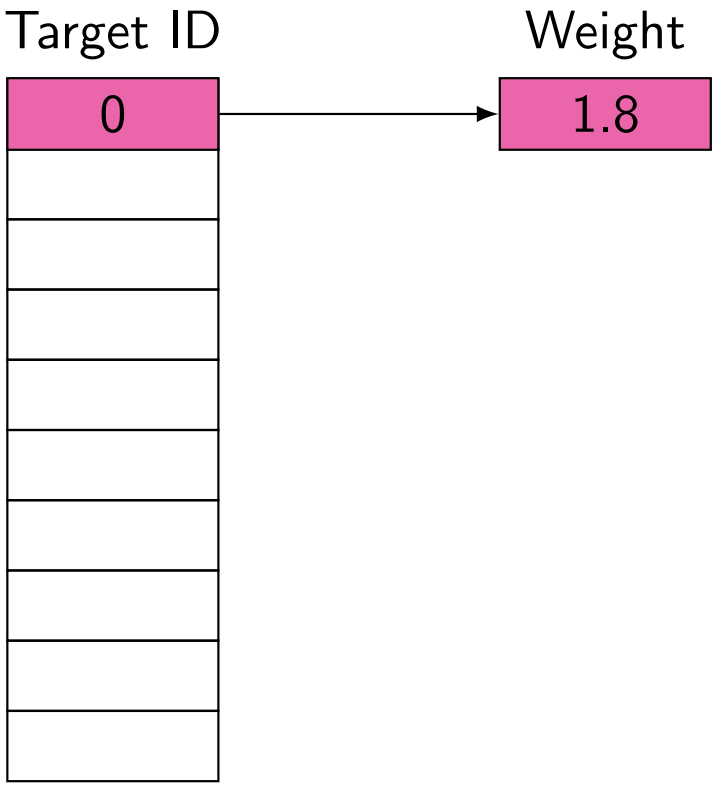
Giraph (Hash Map)

Hash map adjacency list:

Target ID:	0
Weight:	1.8

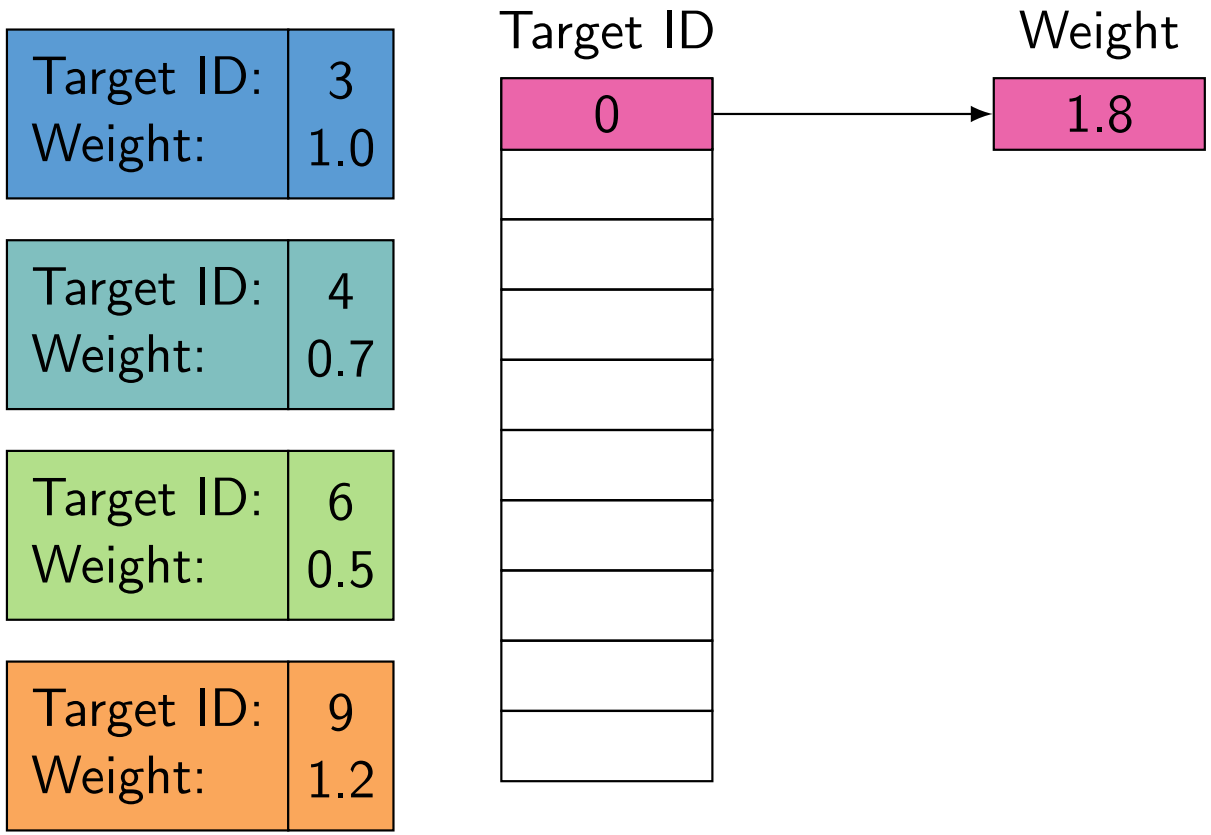
Giraph (Hash Map)

Hash map adjacency list:



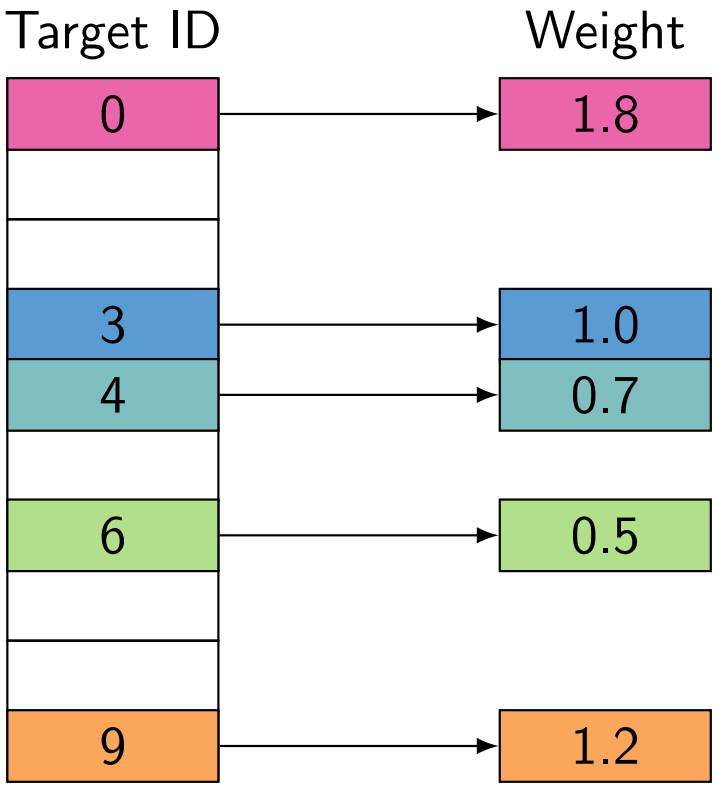
Giraph (Hash Map)

Hash map adjacency list:



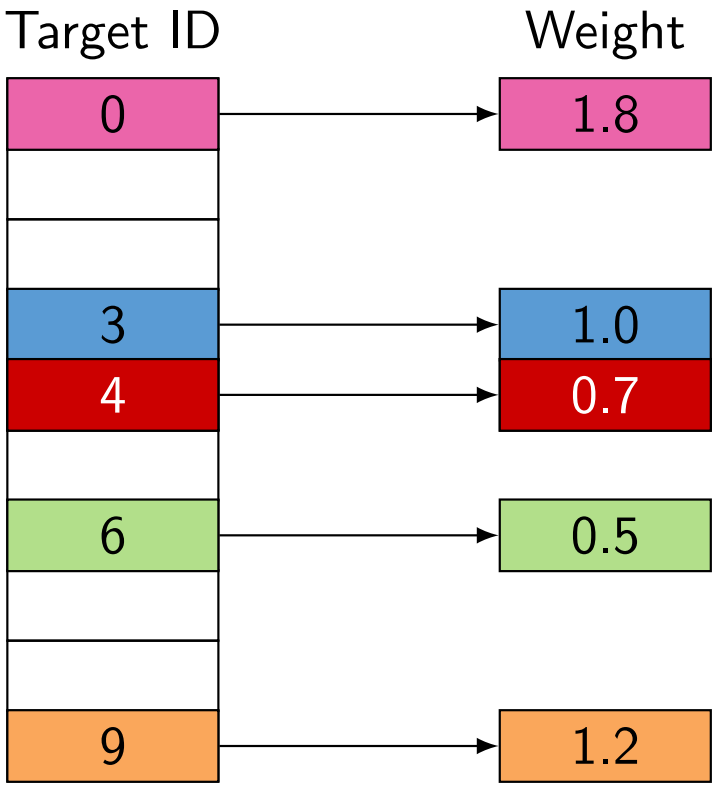
Giraph (Hash Map)

Hash map adjacency list:



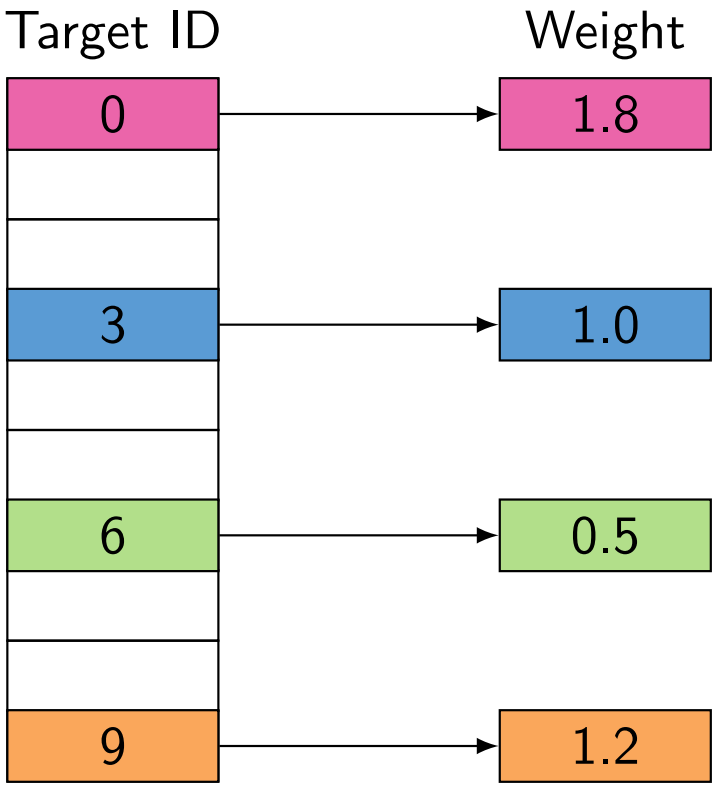
Giraph (Hash Map)

Hash map adjacency list:



Giraph (Hash Map)

Hash map adjacency list:



Giraph (Summary)

Byte array:

- Space efficient. ✓
- Overheads for mutations. ✗

Hash map:

- Less space efficient. ✗
- Efficient for mutations. ✓

Giraph (Summary)

Byte array:

- Space efficient. ✓
- Overheads for mutations. ✗

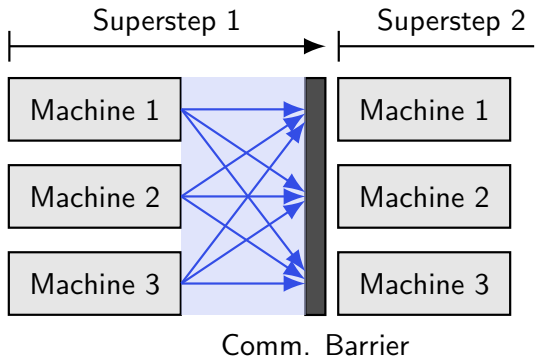
Hash map:

- Less space efficient. ✗
- Efficient for mutations. ✓

What's the tradeoff like?

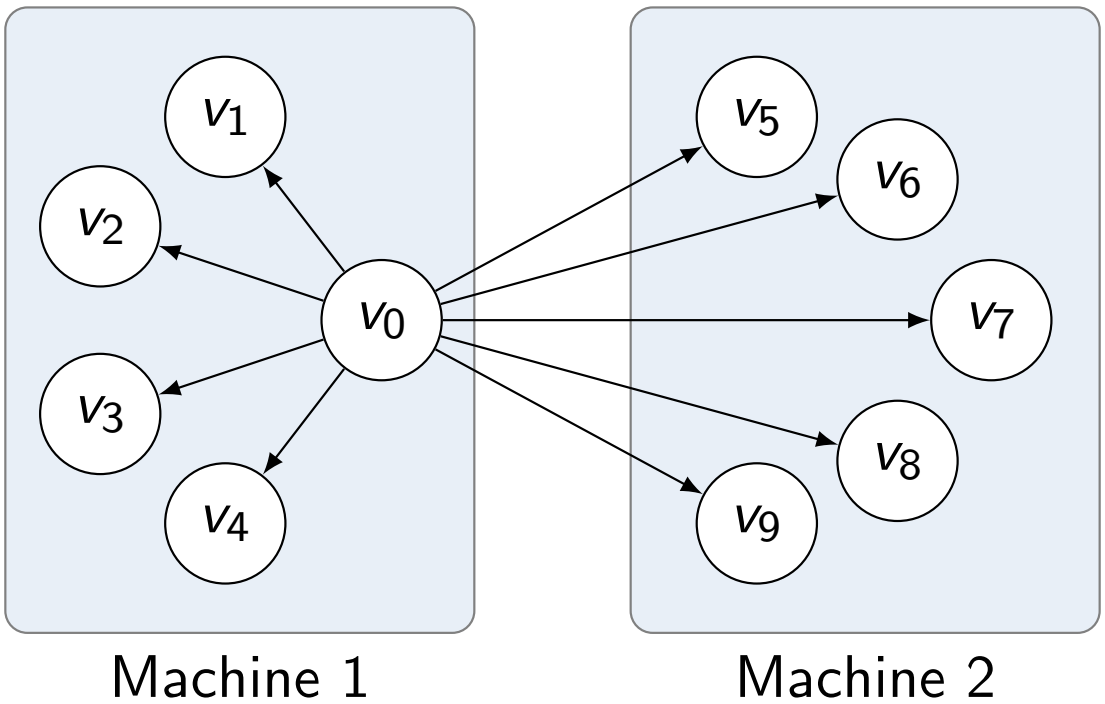
Two optional optimizations:

- LALP.
- Dynamic migration.



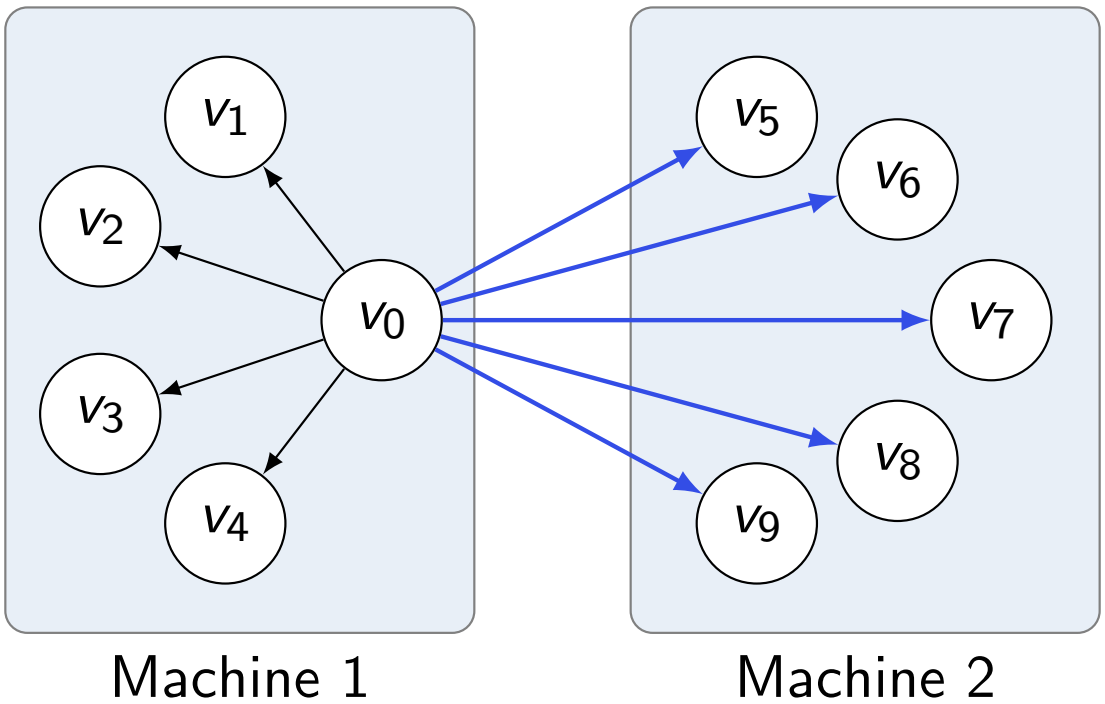
GPS (LALP)

LALP (Large Adjacency List Partitioning):



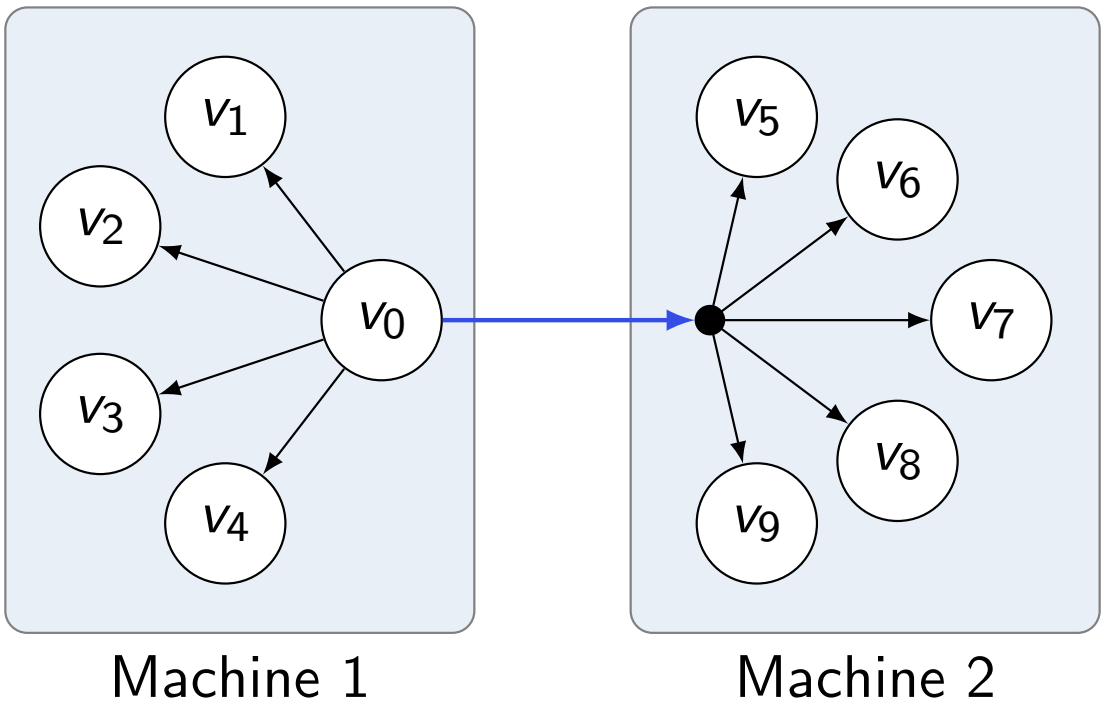
GPS (LALP)

LALP (Large Adjacency List Partitioning):



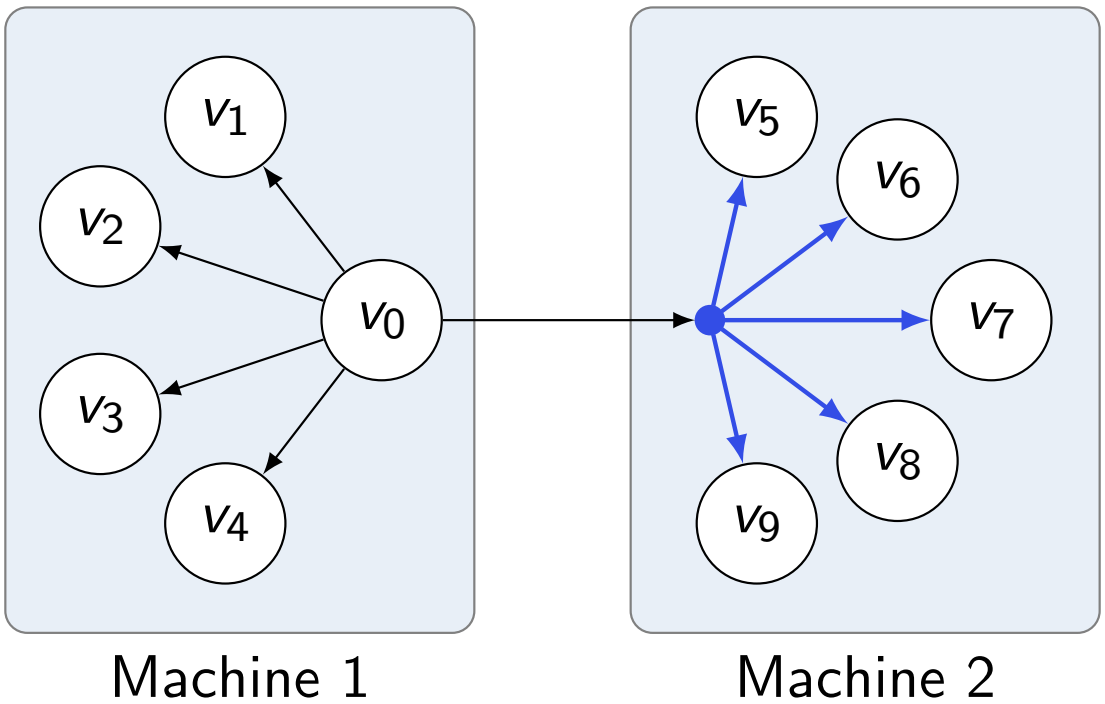
GPS (LALP)

LALP (Large Adjacency List Partitioning):



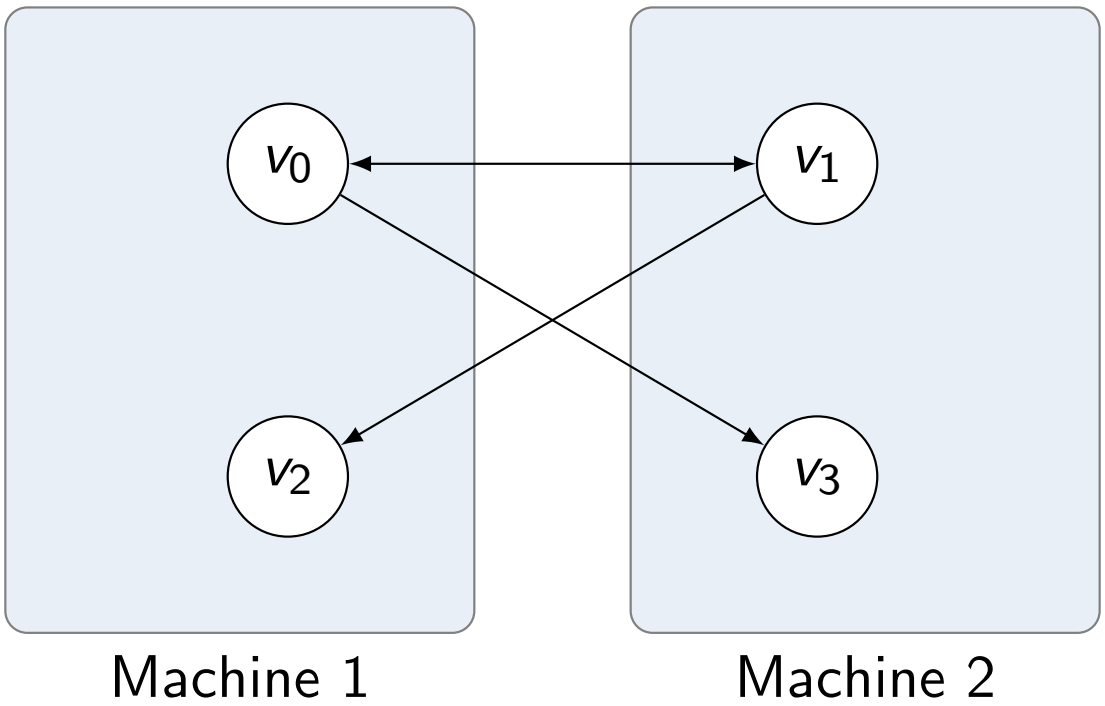
GPS (LALP)

LALP (Large Adjacency List Partitioning):



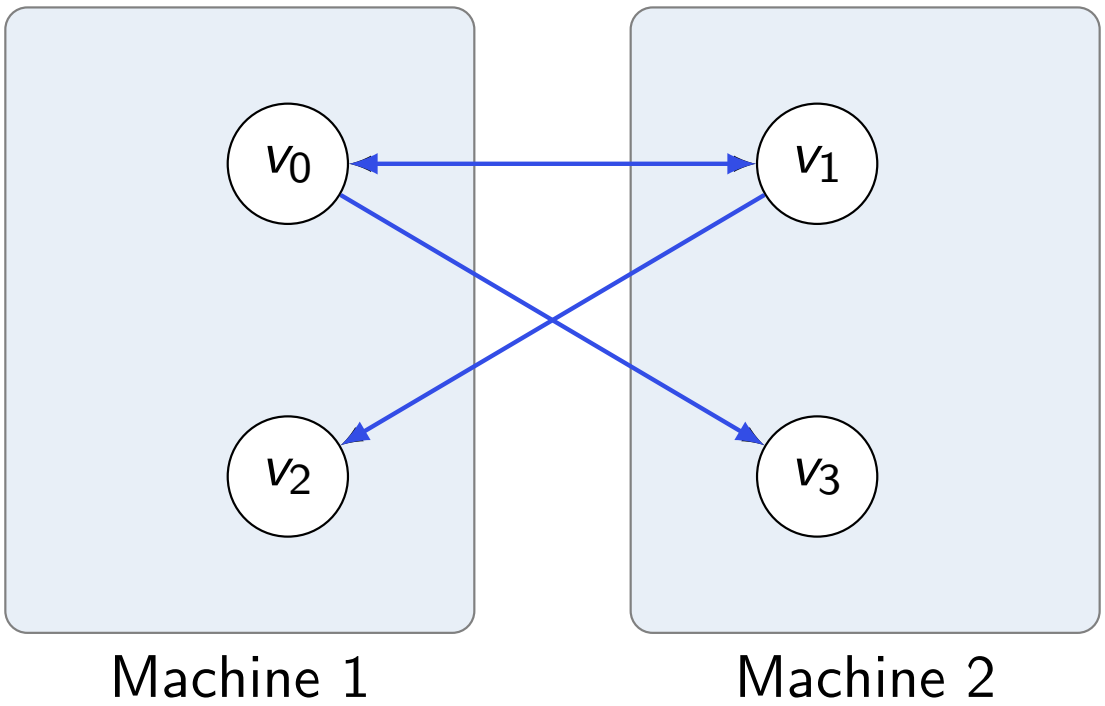
GPS (Dynamic Migration)

Dynamic migration:



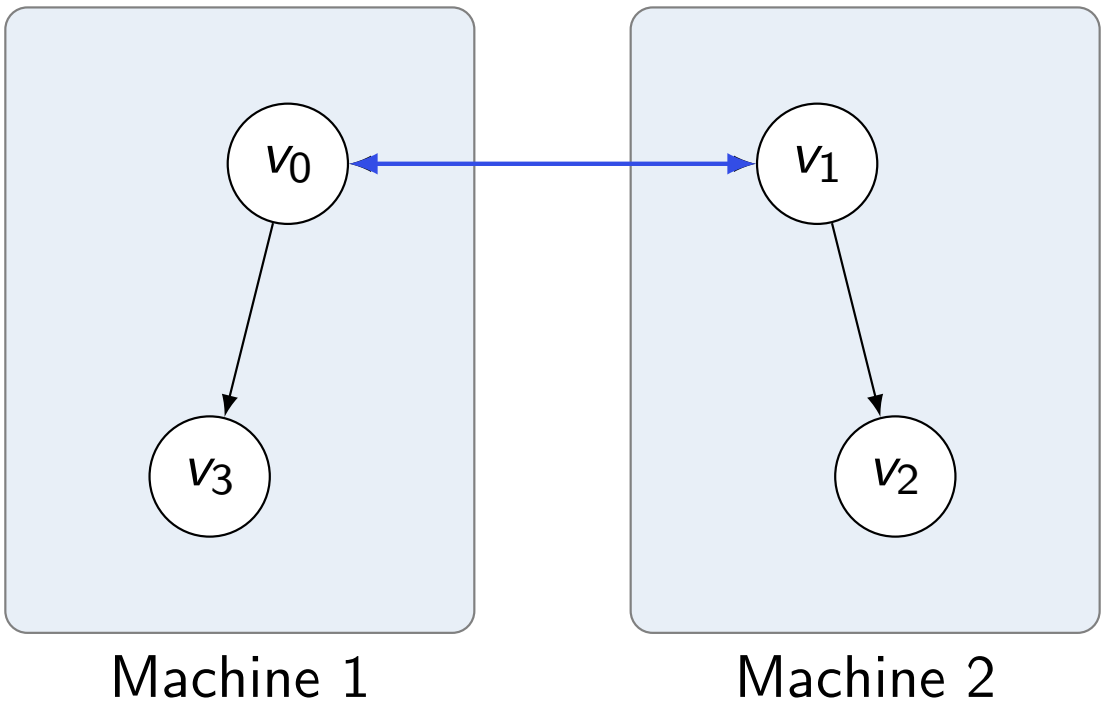
GPS (Dynamic Migration)

Dynamic migration:



GPS (Dynamic Migration)

Dynamic migration:



GPS (Summary)

LALP:

- Reduces network usage. ✓
- Sent messages must be same. ✗

Dynamic migration:

- Reduces network usage. ✓
- Incompatible with DMST. ✗

GPS (Summary)

LALP:

- Reduces network usage. ✓
- Sent messages must be same. ✗

Dynamic migration:

- Reduces network usage. ✓
- Incompatible with DMST. ✗

Do these improve performance?

Mizan

Mizan:

- Lacks built-in system optimizations. ✗
- But competitive against older Giraph 0.1. ✓

Mizan

Mizan:

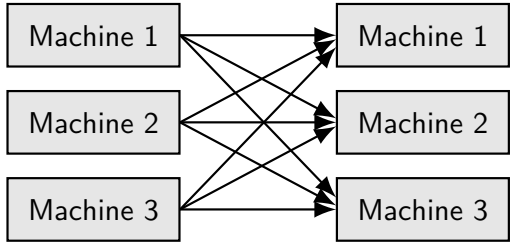
- Lacks built-in system optimizations. ✗
- But competitive against older Giraph 0.1. ✓

How does it compare now?

GraphLab (Asynchronous)

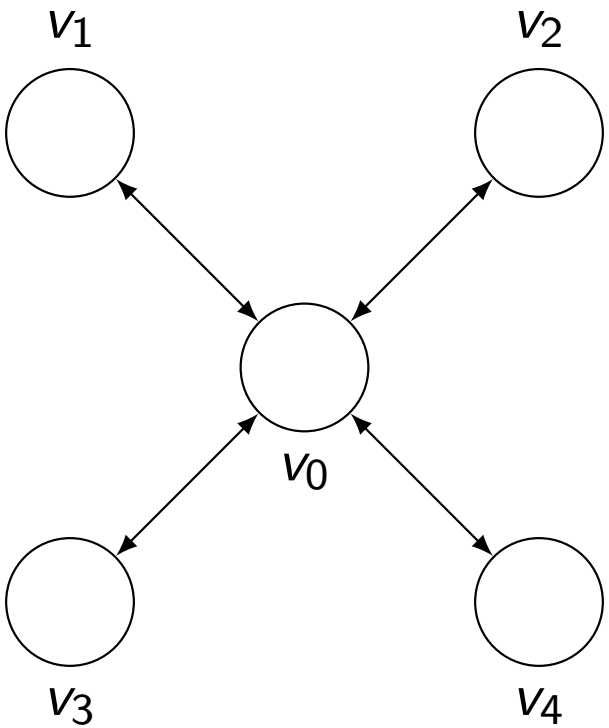
GraphLab features *asynchronous* execution:

- No communication barriers. ✓
- Uses the *most recent* vertex values. ✓



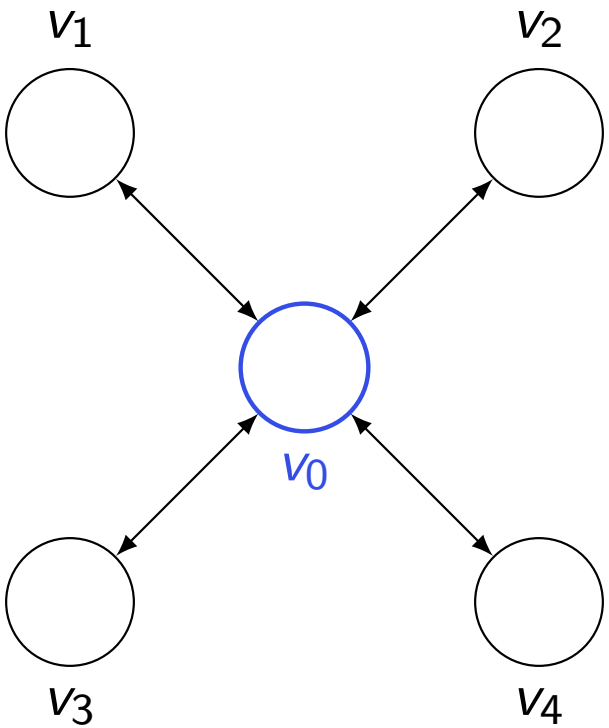
GraphLab (Asynchronous)

Implemented via distributed locking:



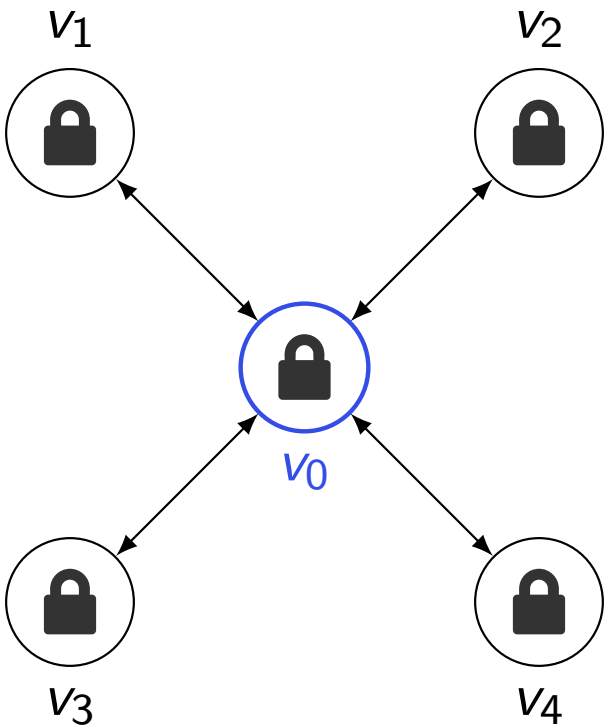
GraphLab (Asynchronous)

Implemented via distributed locking:



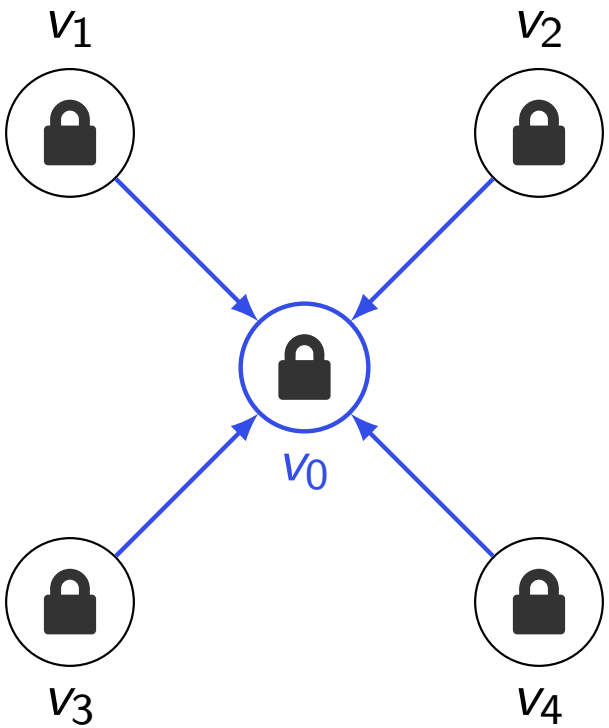
GraphLab (Asynchronous)

Implemented via distributed locking:



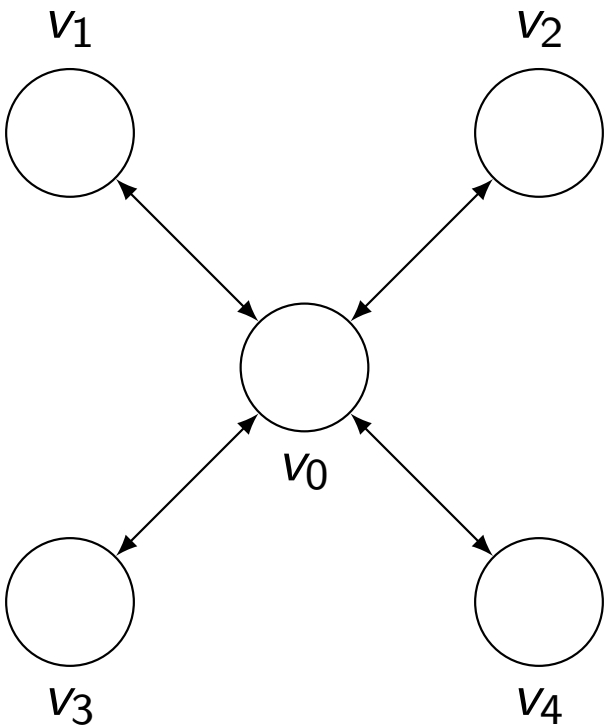
GraphLab (Asynchronous)

Implemented via distributed locking:



GraphLab (Asynchronous)

Implemented via distributed locking:



GraphLab (Asynchronous)

Asynchronous execution:

- No communication barriers. ✓
- Uses the *most recent* vertex values. ✓
- Potential overheads? ✗

GraphLab (Asynchronous)

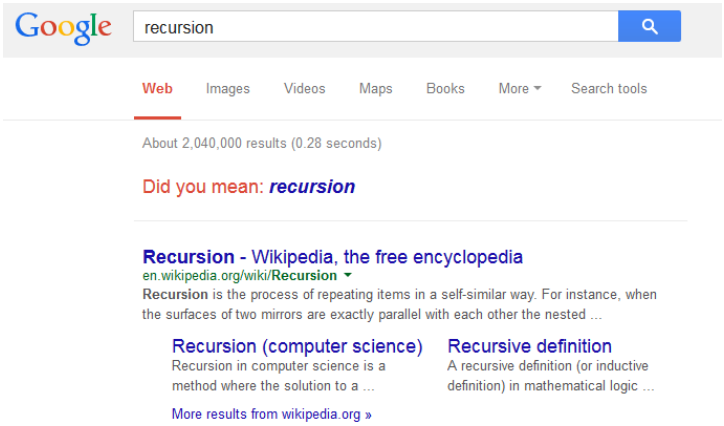
Asynchronous execution:

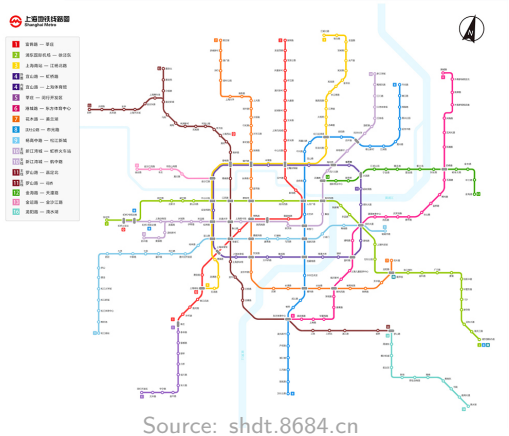
- No communication barriers. ✓
- Uses the *most recent* vertex values. ✓
- Potential overheads? ✗

Performance of sync vs. async?

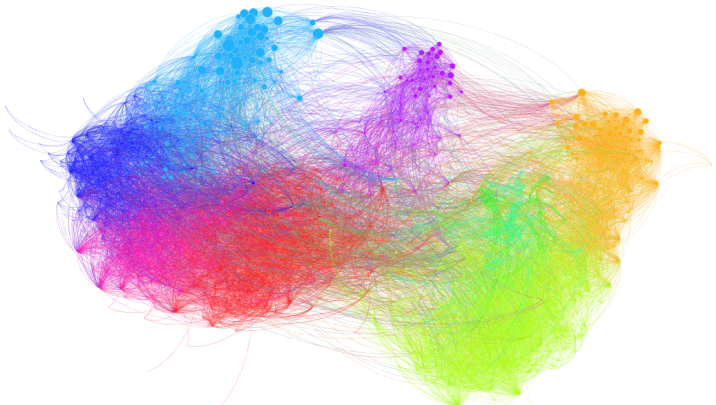
Overview

Tackling Graph Processing Problems

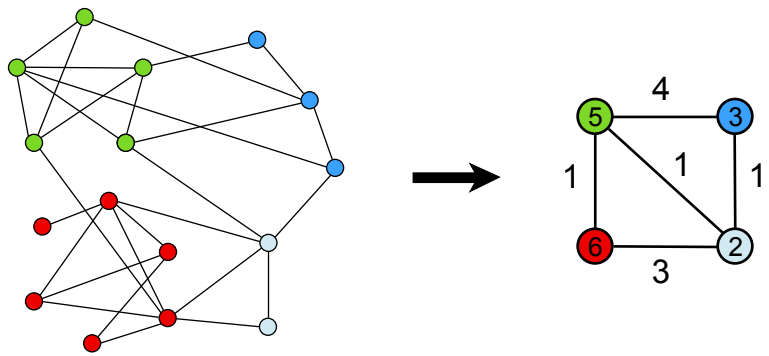




Source: shdt.8684.cn



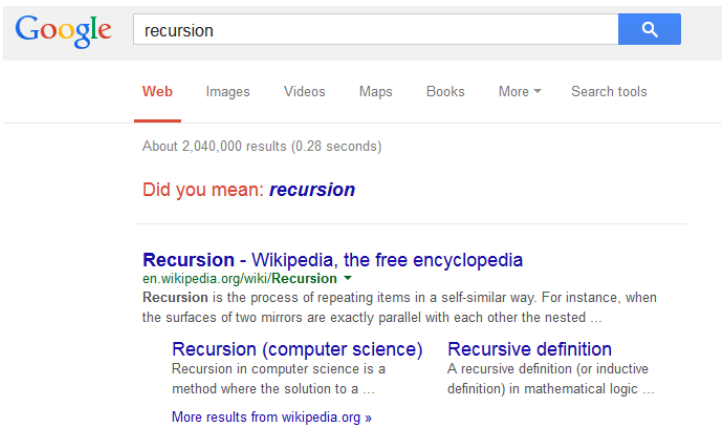
Source: Noguer, Multi-scale graph comparison (2014)



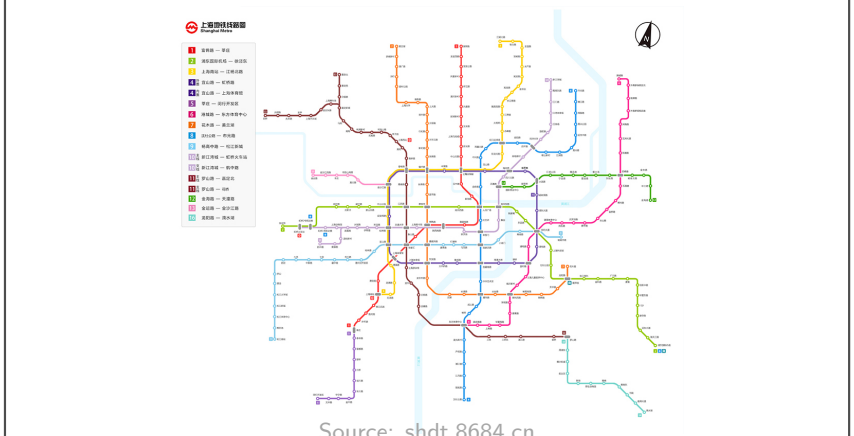
Adapted from: Fortunato, arXiv:0906.0612 (2010)

Tackling Graph Processing Problems

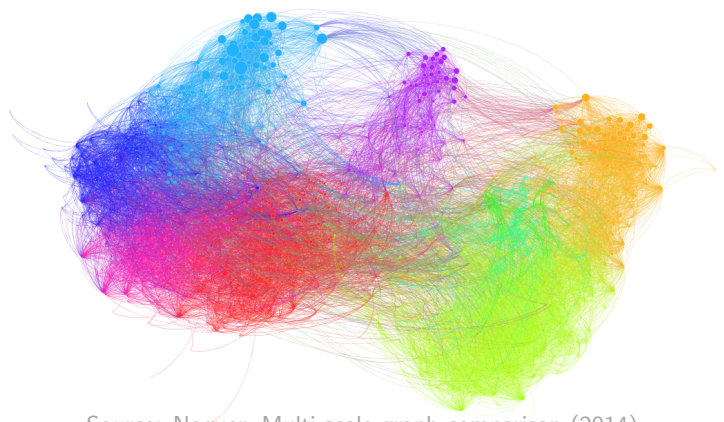
Random Walk



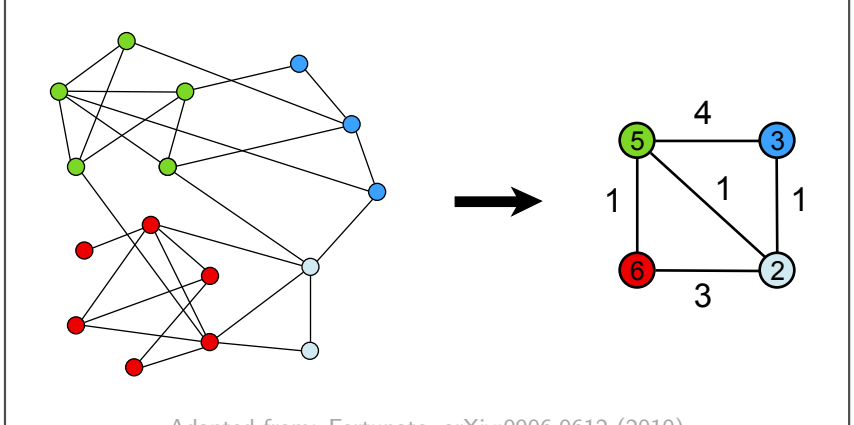
Sequential Traversal



Parallel Traversal



Graph Mutation



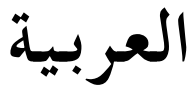
Algorithms

Category	Algorithm
Random walk	PageRank
Sequential traversal	SSSP (Single Source Shortest Path)
Parallel traversal	WCC (Weakly Connected Components)
Graph mutation	DMST (Distributed Minimum Spanning Tree)

Datasets

LJ

OR



AR

TW

UK

Datasets

LJ

OR

العربية

AR

TW

$|E| = 1.46\text{B}$

UK

$|E| = 3.73\text{B}$

Datasets

LJ

OR

AR

TW

$|E| = 1.46\text{B}$

UK

$|E| = 3.73\text{B}$

Datasets are:

- Stored on HDFS as text files.
- Loaded via random hash partitioning.

Evaluation Metrics

- **Time:** *total* time = *setup* time + *computation* time
- **Memory:** *maximum* memory usage (across all machines)
- **Network:** *total* network usage (summed over all machines)

Setup

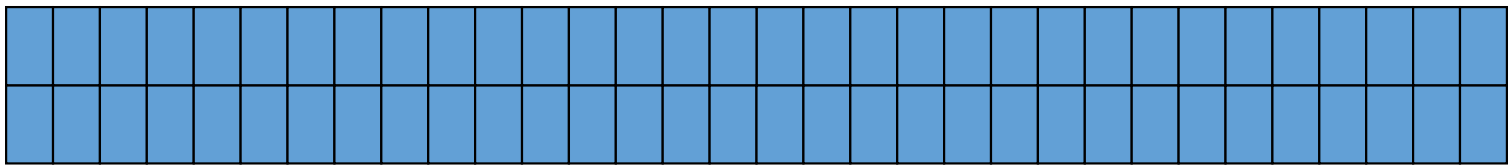
EC2 Instances

Setup

EC2 Instances

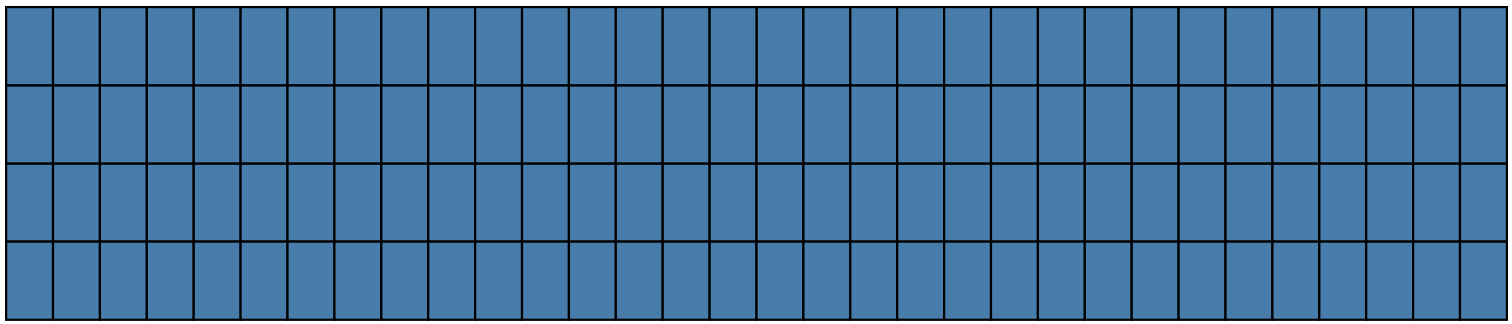
Setup

EC2 Instances



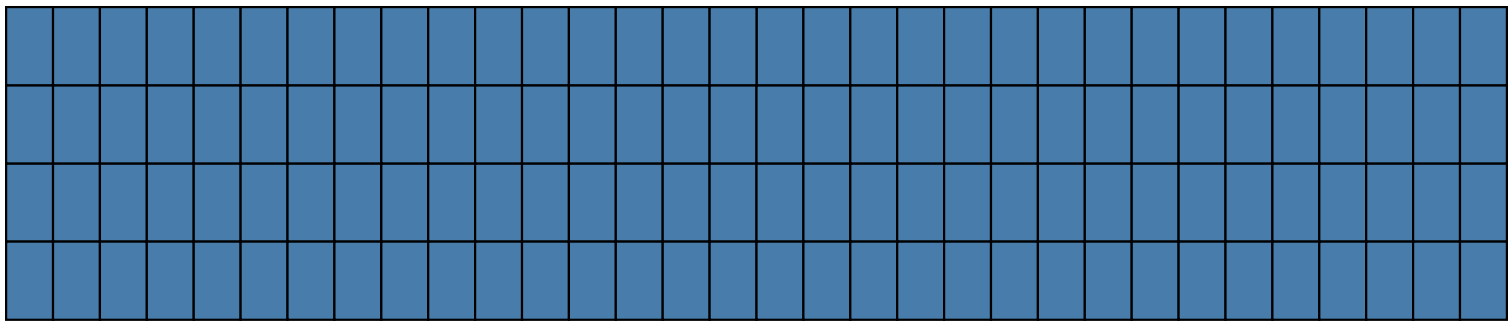
Setup

EC2 Instances



Setup

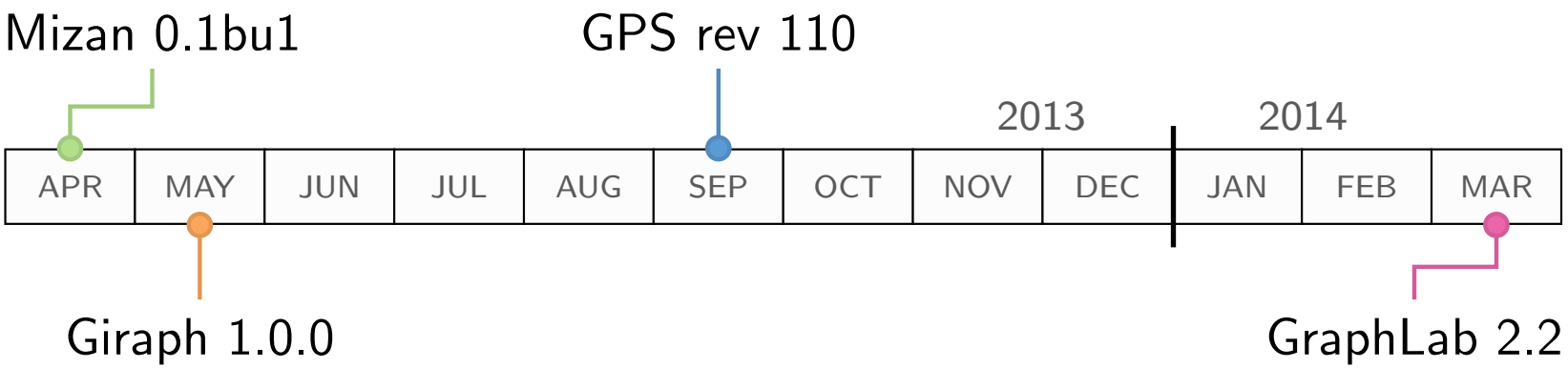
EC2 Instances



- m1.xlarge: 4 vCPUs, 15GB memory, Ubuntu 12.04

Setup

- m1.xlarge: 4 vCPUs, 15GB memory, Ubuntu 12.04



Overview

Read our paper!

VLDB 2014

For memory, we define per-machine memory usage as the difference between the maximum and minimum memory used by a single machine during an experiment. This excludes the background usage of Hadoop and OS processes, which is typically between 200 to 300MB. We focus on *maximum memory usage*, the maximum per-machine usage across all worker machines. This gives the minimum memory resources all machines need to run an experiment without failure.

Similarly, per-machine network usage is the total number of bytes sent or received by a single machine for an experiment. We focus on *total network usage*, the sum of per-machine usage across all worker machines, as it best enables a high-level comparison of each system's network I/O. Additionally, we distinguish between total outgoing (sent) and total incoming (received) network usage.

Lastly, using different datasets and number of machines enables us to investigate the *scalability* of each system: how performance scales with the same graph on more machines or with larger graphs on the same number of machines.

To track these metrics, we use the total and setup times reported by all systems. For network usage, some systems have built-in message counters but they all differ. Message count is also a poor measure of network traffic, as messages can be of different sizes (such as in DMST) and header sizes are ignored. Further, the notion of messages is ill-defined for GraphLab's asynchronous mode. For these reasons, we use `/proc/net/dev` to track the total bytes sent and received at each machine, before and after every experiment.

For more fine-grained statistics, and to compute memory usage, we rely on one-second interval reports from `sar` and `free`. These are started locally on both the master and worker machines to record CPU, network, and memory utilization in a decentralized manner. They are started before and killed after each experiment to ensure a small but constant overhead across all experiments. Finally, Giraph 1.0.0 has a bug that prevents proper clean up of Java processes after a successful run. We solve this by killing the offending processes after each run.

For each experiment, we perform five runs and report both the mean and 95% confidence intervals for computation time, setup time, maximum memory usage, and total incoming network usage. We show only incoming network I/O as it is similar to total outgoing network I/O and exhibits identical patterns.

6. EXPERIMENTAL RESULTS

In this section, we present and analyze our experimental results. Due to space constraints and in the interests of readability, we present only a subset of all our experimental data. In particular, we do not show data for the master, since its memory and network usage are smaller and less revealing of system performance. All data and plots, along with our code and EC2 images, are available online.[4]

Table 4: Performance for `AR`, `TW`, `UK`.

	Computation	Setup	Total Time	Memory	Network
No Mutations					
Gi-BA	★★★☆	★★★★	★★★★	★★★☆	★★★★
Gi-HM	★★★☆	★★★☆	★★★☆	★★☆☆	★★★★
GPS	★★★☆	★★☆☆	★★☆☆	★★★★	★★☆☆
Mizan	★☆☆☆	★☆☆☆	★☆☆☆	★☆☆☆	★★★☆
GL-S	★★★★	★★★☆	★★★★	★★★☆	★★★☆
GL-A	★☆☆☆	★★★☆	★☆☆☆	★★☆☆	★☆☆☆
With Mutations (DMST)					
Gi-BA	★☆☆☆	★★★★	★☆☆☆	★★☆☆	★★★★
Gi-HM	★★★★	★★★★	★★★★	★★☆☆	★★★★
GPS	★★★★	★☆☆☆	★★☆☆	★★★★	★☆☆☆

6.1 Summary of Results

A relative ranking of system performance, for computation, setup, and total time, maximum memory usage, and total network I/O, are presented for non-mutation and mutation (DMST) algorithms in Table 4. System names are abbreviated as Gi-BA and Gi-HM for Giraph byte array and hash map respectively; GPS for GPS with no optional optimizations; and GL-S and GL-A for GraphLab's synchronous and asynchronous modes. We exclude GPS's other modes as they provide little performance benefits. We focus on the larger graphs (`AR`, `TW`, and `UK`) and summarize our findings for `LJ` and `OR` in Section 6.6.

For each performance attribute, we provide a 4 star relative ranking, with 1 for poor performance, 2 for suboptimal, 3 for good, and 4 for excellent. The ranking of a system is roughly an average of its performance across algorithms.

Overall, for the non-mutation algorithms tested, Giraph and GraphLab are close: we recommend Giraph byte array over GraphLab for very large graphs on a limited number of machines, but GraphLab's synchronous mode over Giraph otherwise. In each case, the recommended system has the best all-around performance. When mutations are required, we recommend Giraph hash map over byte array if memory is not a constraint. If memory is the primary constraint, then GPS is the best option for both mutation and non-mutation algorithms.

Next, we discuss results for each system, focusing on the larger graphs, before highlighting our findings for the `LJ` and `OR` graphs.

6.2 Giraph

Compared to GPS, Giraph's computation times are longer for PageRank and comparable or shorter for SSSP and WCC (Figure 2). Since Giraph's setup times are much shorter than GPS's, especially on 64 and 128 machines, Giraph is faster overall: up to 3× shorter total time on PageR-

GitHub

Code, results, wiki, and EC2 images:

`http://xvz.github.io/graph-processing`

Findings and Lessons Learned (Giraph)

1. Graph storage should be memory and mutation efficient.

No Mutations

	Time	Memory
Byte array	✓	✓
Hash map	✗	✗

With Mutations (DMST)

	Time	Memory
Byte array	✗✗	✓
Hash map	✓	✗

Findings and Lessons Learned (GPS)

2. LALP and dynamic migration provide little gains.
   - **Time:** negative to no improvements. ✗
   - **Memory:** large overheads on large graphs. ✗
   - **Network:** little to no improvements. ✓

Findings and Lessons Learned (Mizan)

3. Message *processing* optimizations are very important.

	Computation	Setup	Memory	Network
		No Mutations		
Giraph (byte array)	★★★☆	★★★★	★★★☆	★★★★
Giraph (hash map)	★★★☆	★★★☆	★★☆☆	★★★★
GPS (none)	★★★☆	★★☆☆	★★★★	★★☆☆
Mizan	★☆☆☆	★☆☆☆	★☆☆☆	★★★☆
GraphLab (sync)	★★★★	★★★☆	★★★☆	★★★☆
GraphLab (async)	★☆☆☆	★★★☆	★★☆☆	★☆☆☆

Findings and Lessons Learned (GraphLab)

4. Distributed locking for asynchronous execution is not scalable.
   - Performance degrades as more machines are used. ✗
   - Due to: lock contention, termination scheme, lack of message batching.

Findings and Lessons Learned

5. Giraph scales better across graphs;
   GraphLab scales better across more machines.

Findings and Lessons Learned

5. Giraph scales better across graphs;
   GraphLab scales better across more machines.

64 machines	TW	UK
Giraph (byte array)	5.8GB	7.0GB
GraphLab (sync)	4.5GB	14GB

Findings and Lessons Learned

5. Giraph scales better across graphs;
   GraphLab scales better across more machines.

64 machines	TW	UK
Giraph (byte array)	5.8GB	7.0GB
GraphLab (sync)	4.5GB	14GB

TW	16 machines	128 machines
Giraph (byte array)	8.5GB	5.8GB
GraphLab (sync)	11GB	3.3GB

Comparison Summary

	Computation	Setup	Memory	Network
No Mutations				
Giraph (byte array)	★★★☆	★★★★	★★★☆	★★★★
Giraph (hash map)	★★★☆	★★★☆	★★☆☆	★★★★
GPS (none)	★★★☆	★★☆☆	★★★★	★★☆☆
Mizan	★☆☆☆	★☆☆☆	★☆☆☆	★★★☆
GraphLab (sync)	★★★★	★★★☆	★★★☆	★★★☆
GraphLab (async)	★☆☆☆	★★★☆	★★☆☆	★☆☆☆
With Mutations (DMST)				
Giraph (byte array)	★☆☆☆	★★★★	★★☆☆	★★★★
Giraph (hash map)	★★★★	★★★★	★★☆☆	★★★★
GPS (none)	★★★★	★☆☆☆	★★★★	★☆☆☆

Summary

- Comprehensive experimental comparison of four graph processing systems, with scale.
- System optimizations are critical, but some optimizations can degrade performance.
- Performance and scalability depend heavily on metric used: there is no clear winner.

Summary

- Comprehensive experimental comparison of four graph processing systems, with scale.
- System optimizations are critical, but some optimizations can degrade performance.
- Performance and scalability depend heavily on metric used: there is no clear winner.

Summary

- Comprehensive experimental comparison of four graph processing systems, with scale.
- System optimizations are critical, but some optimizations can degrade performance.
- Performance and scalability depend heavily on metric used: there is no clear winner.