SAGA-Bench: Software and Hardware Characterization of StreAming Graph Analytics Workloads

Abanti Basak, Jilan Lin, Ryan Lorica, Xinfeng Xie, Zeshan Chishti*, Alaa Alameldeen*, and Yuan Xie

University of California, Santa Barbara
**Intel*

UC Santa Barbara
Scalable Energy-efficient Architecture Lab

Executive Summary

Streaming graph analytics and its unique challenges

SAGA-Bench: an open-source benchmark for streaming graphs

Software-level characterization of different data structures and compute models

Architecture-level characterization of graph update and graph compute phases

Section I

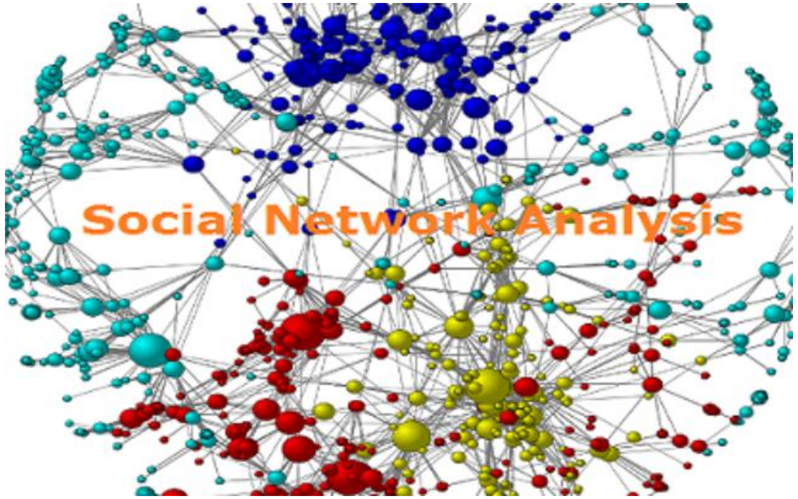
Streaming graph analytics and its unique challenges

Application Domains of Streaming Graphs

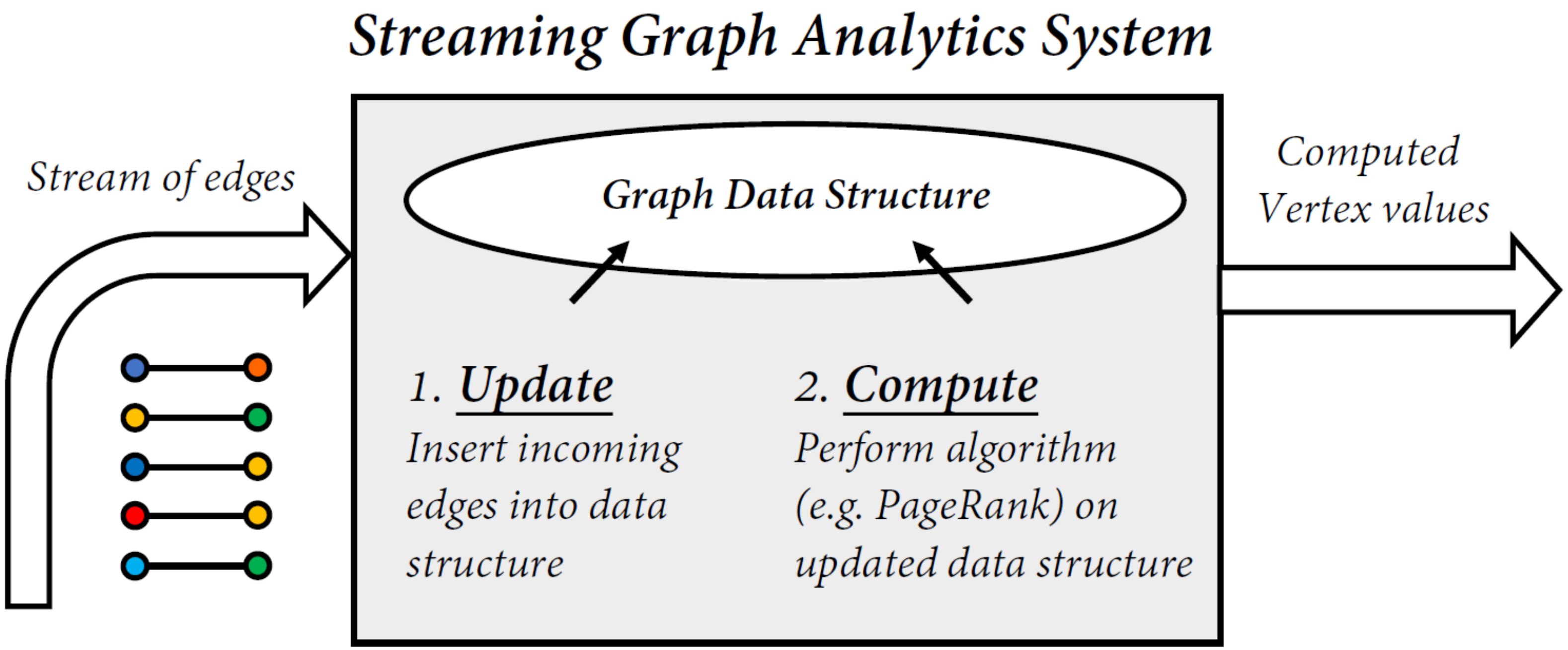
Financial fraud detection

Recommender systems

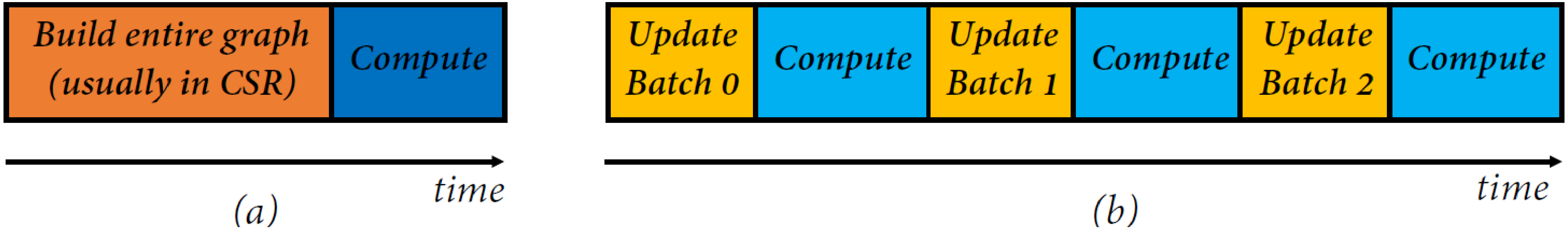
Social Network Analysis



Streaming Graph Analytics Overview



Difference Between Static and Streaming Graphs



STATIC

- Build graph once, compute again and again
- Optimization goal: execution time of compute phase
- Graph update is a fixed one-time overhead

STREAMING

- Repeated update and compute on batches of incoming edges
- Optimization goal: real-timeliness, i.e., low batch processing latency
- Graph update lies on the critical path

Shortcomings of Prior Software Work

Aspen *(PLDI 2019)*

GraphOne *(USENIX FAST 2019)*

Stinger *(HPEC 2012)*

KickStarter *(ASPLOS 2017)*

Kineograph *(EuroSys 2012)*

GraPU *(SoCC 2018)*

Degree-Aware Hashing *(IPDPSW 2016)*

GraphTinker *(IPDPS 2019)*

Multiple stand-alone streaming graph systems but lack of systematic study of the software techniques (data structures and compute models) proposed across these systems

Shortcomings of Prior Architecture Work

Graphicionado *(MICRO 2016)*

HATS *(MICRO 2018)*

GraphP *(HPCA 2018)*

Tesseract *(ISCA 2015)*

PHI *(MICRO 2019)*

Droplet *(HPCA 2019)*

GraphQ *(MICRO 2019)*

Multiple papers on static graph computation but streaming graphs remain unexplored at architecture level due to:

- Immature software techniques
- Lack of open-source benchmarks

This Work

Creates SAGA-Bench, an open-source benchmark, and performs systematic software and hardware characterization of streaming graph analytics workloads

Section II

SAGA-Bench: an open-source benchmark for streaming graphs

SAGA-Bench Overview

Benchmark in C++ which puts together different data structures and compute models for streaming graph analytics on the same platform for systematic characterization

GitHub repo: https://github.com/abasak24/SAGA-Bench

Scope of SAGA-Bench

Software Studies: Common platform for performance analysis of software techniques such as different data structures and compute models

Architecture-level studies: Open source tool for studying architecture-level bottlenecks in streaming graph applications

Extensible: The API of SAGA-Bench is general enough to accommodate future software techniques

SAGA-Bench Contents

Data Structures (all support multithreading):

- Stinger
- Degree-Aware Hashing (DAH)
- Adjacency List (shared-style multithreading) (AS)
- Adjacency List (chunked-style multithreading) (AC)

Compute Models:

- Incremental
- From scratch

Implemented Algs (all support multithreading):

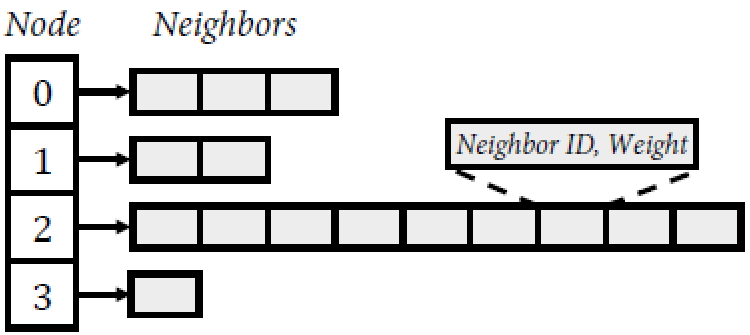
- Breadth First Search (BFS)
- Connected Components (CC)
- Max Computation (MC)
- PageRank (PR)
- Single Source Shortest Path (SSSP)
- Single Source Widest Path (SSWP)

4 data structures + 6 x 2 algorithms

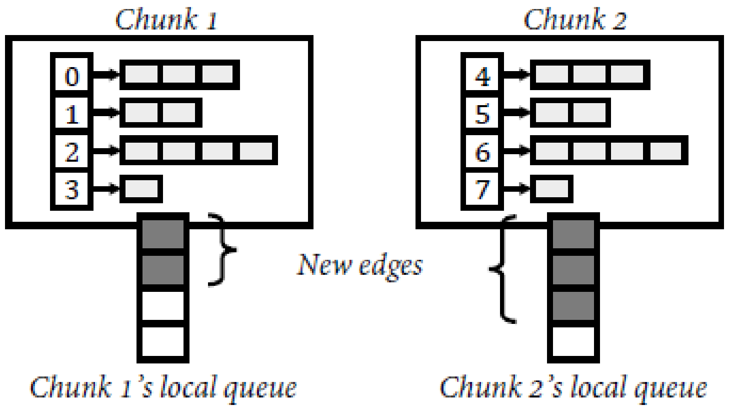
Data Structures

Graph Update Mechanism

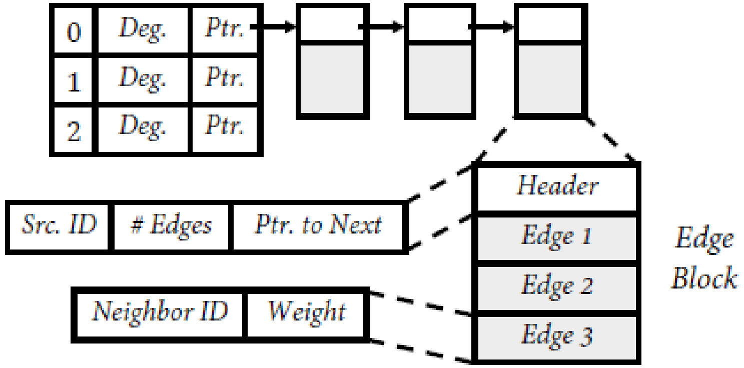
Intra-node Parallelism



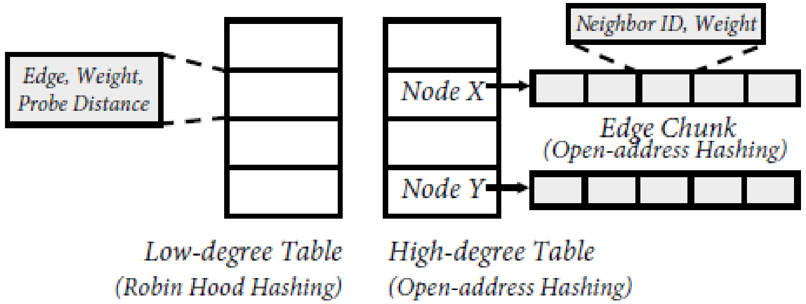
Shared adjacency list (AS)



Chunked adjacency list (AC)



Stinger



Degree-Aware Hashing (DAH)

Multithreading Technique

Traversal Mechanism

Compute Models

Recomputation From scratch (FS)		Incremental Computation (INC)
Update new edges	Batch 0	Update new edges
Reset vertex properties to initial values		Reset vertex properties to initial values
Perform algorithm		Perform algorithm
Update new edges	Batch 1	Update new edges
Reset vertex properties to initial values		**Reuse old computed vertex values from previous batch + compute starting from affected vertices**
Perform algorithm		Perform algorithm

time

Section III

Software-level characterization of different data structures and compute models

Experimental Setup

Platform

- Intel Xeon Gold 6142 (Skylake) server
- Dual-socket, 64 total HW execution threads
- 32KB private L1, 1MB private L2, 22MB shared LLC
- 768GB DRAM, 128GB/s memory BW per socket
- 136.2 GB/s inter-socket communication

Methodology

- Shuffle datasets and stream batches of 500K edges
- Three representative data points P1, P2, P3 for early, middle, and final stages
- Averages with 95% confidence intervals

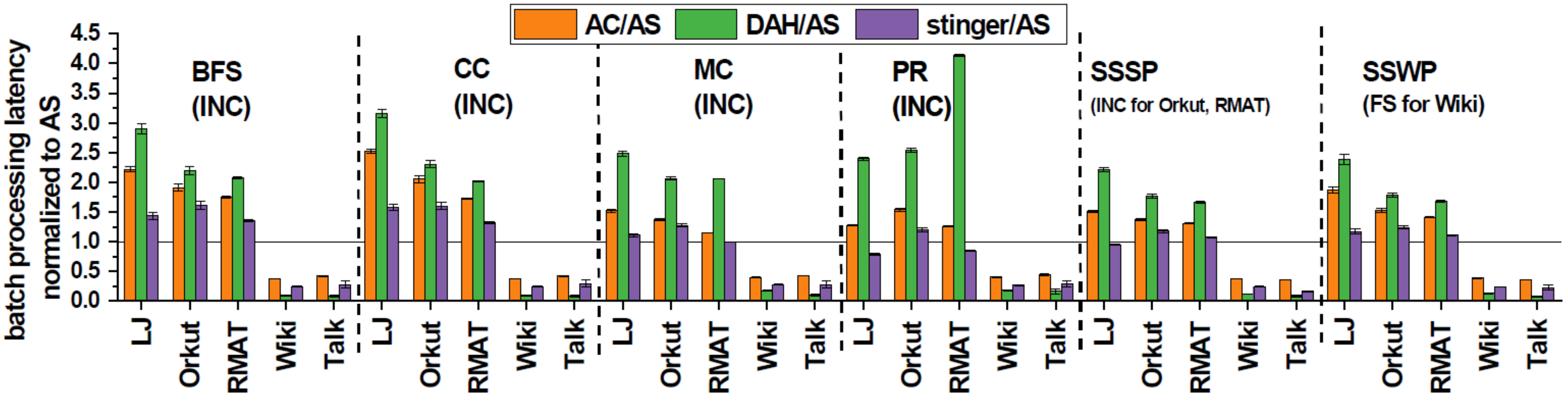
Datasets

Dataset	vertices	edges	batchCount
Livejournal (LJ)	4,847,571	68,993,773	138
Orkut	3,072,441	117,185,083	235
RMAT	32,118,308	500,000,000	1000
wiki-topcats (Wiki)	1,791,489	28,511,807	58
wiki-talk (Talk)	2,394,385	5,021,410	11

Software Profiling Overview

- Which data structure is the best?
- Which compute model is the best?
- What proportions of the batch processing latency do update and compute phases occupy?

Best Data Structure depends on Per-Batch Degree Distribution of the Graph



worst → **best**

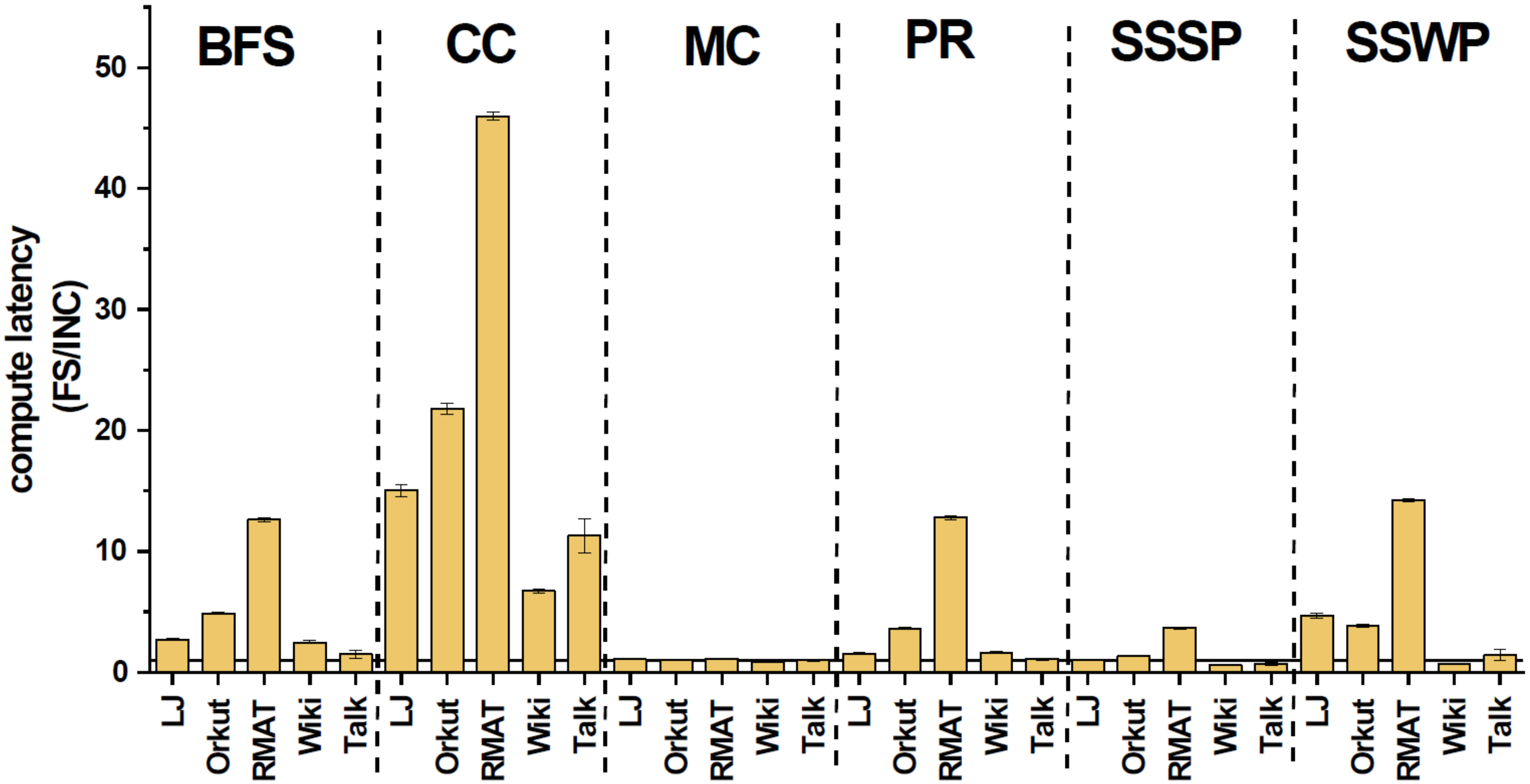
LJ, Orkut, RMAT: DAH > AC > Stinger > AS

Wiki, Talk: AS > AC > Stinger > DAH

Dataset	Entire Dataset		One Batch	
	Max In-degree	Max Out-degree	Max In-degree	Max Out-degree
LJ	13906	20293	106	147
Orkut	33313	33313	144	144
RMAT	8016	7997	10	10
Wiki	**238040**	3907	**4174**	70
Talk	3311	**100022**	330	**9957**

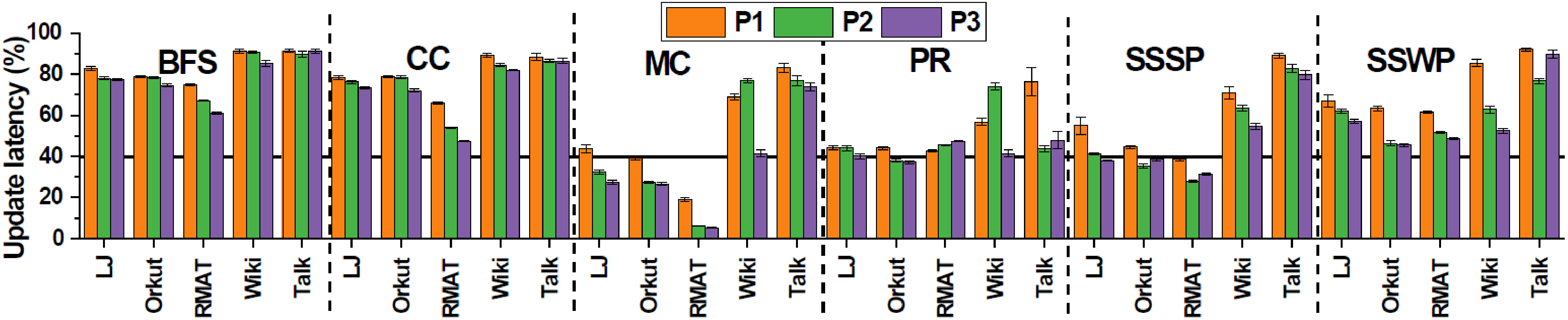
Per-batch degree distribution of LJ, Orkut, RMAT is short-tailed (low imbalance).
Per-batch degree distribution of Wiki, Talk is heavy-tailed (high imbalance).

Larger Graphs Benefit More from Incremental Compute Model



In general, RMAT, the largest dataset, benefits the most from incremental compute model

Batch Processing Latency Breakdown



Update phase is non-trivial in streaming graph analytics.
More than 40% latency comes from update phase in many cases.

Section IV

Architecture-level characterization of graph update and graph compute phases

- Compute Model: Incremental
- Data structure: Adjacency List (AS) for LJ, Orkut, Rmat (**STail**)
  Degree-Aware Hashing (DAH) for Wiki, Talk (**HTail**)
- Profiling tool: Intel Processor Counter Monitor (PCM)

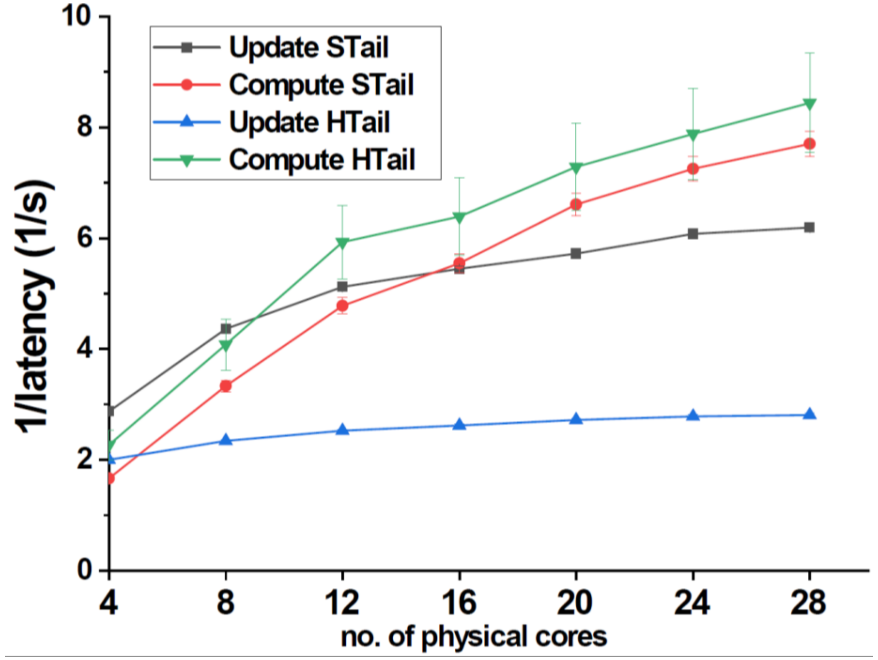
Architecture Profiling Overview

- How do update and compute phases utilize different architecture resources?

- What influences the architecture resource utilization of the update phase?

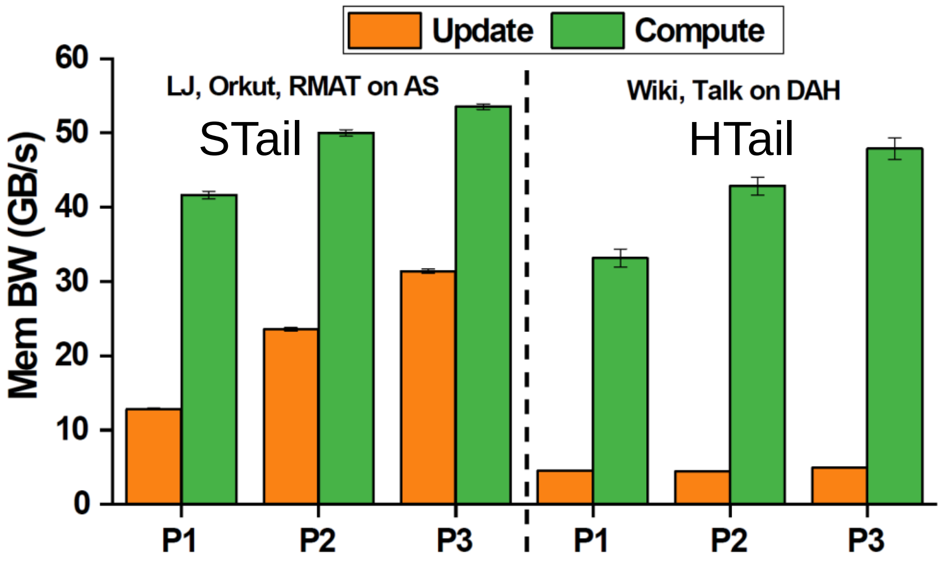
Update Phase Shows Lower Utilization of Resources

Core scaling



Update: good scalability up to ~8-12 cores
Compute: good scalability up to ~20 cores

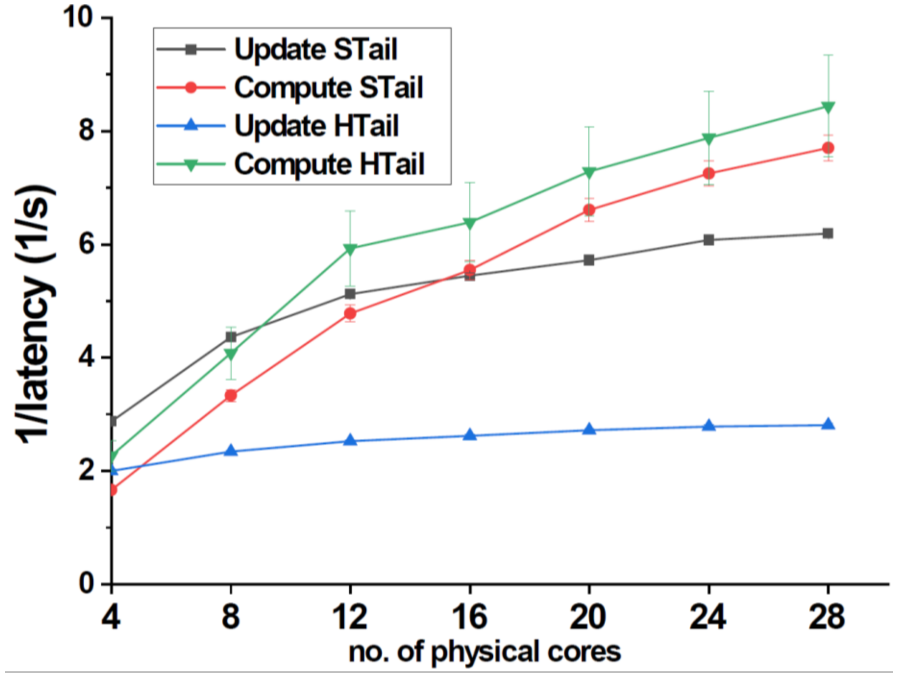
Memory BW utilization



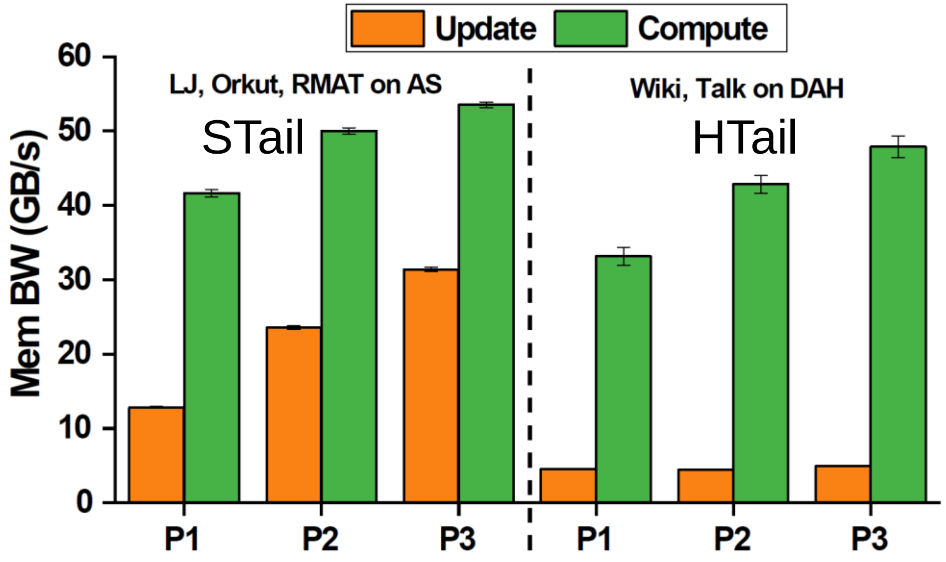
Update uses lower memory BW than Compute

Structure of Graph's Batches Influences Resource Utilization of Update Phase

Core scaling



Memory BW utilization



HTail Update: poor scalability beyond 4-8 cores

STail Update: 13-32GB/s
HTail Update: ~5GB/s

Conclusions

- Streaming graph analytics is important in many application domains and possesses unique challenges. However, there is a lack of systematic software and hardware studies.

- **Contribution 1**: SAGA-Bench, an open-source benchmark.

- **Contribution 2**: Systematic software characterization to provide insights on the best data structure, best compute model, and latency breakdown.

- **Contribution 3**: Architecture-level characterization to study how the update and compute phases utilize different architecture resources.