Analyzing Graph Structure in Streaming Data with STINGER

Jason Riedy, Robert C. McColl, David Ediger, and David A. Bader with Anita Zakrzewska and Oded Green
Georgia Institute of Technology

28 February 2013

Georgia Tech College of Computing
Computational Science and Engineering

Outline

Background

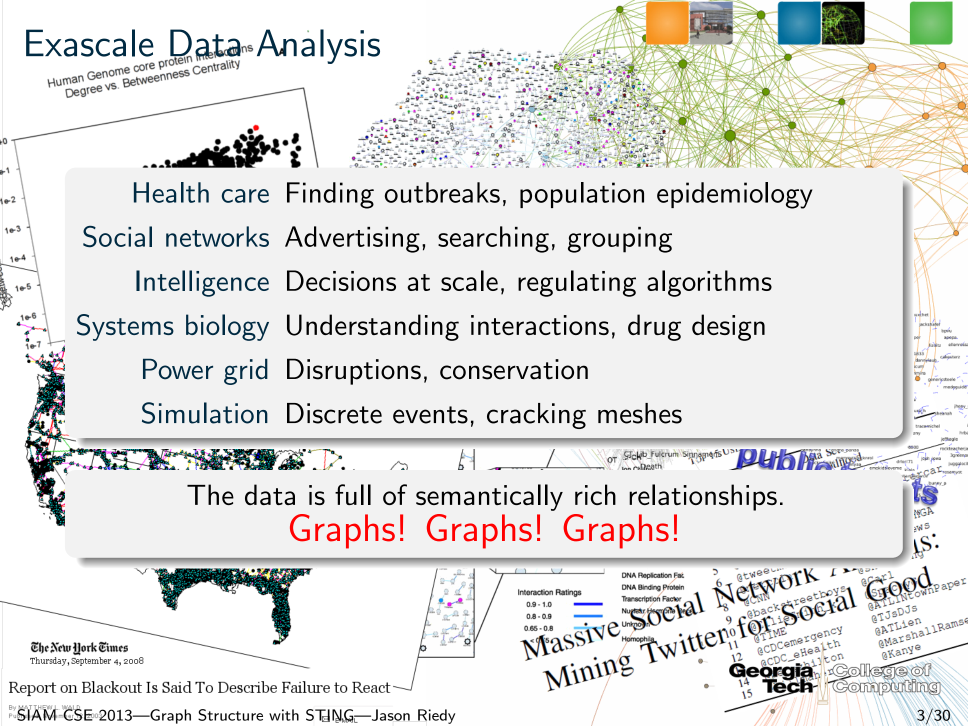
- Where graphs appear... (hint: everywhere)
- Data volumes and rates of change
- Why analyze data streams?

Technical Bits

- How to analyze streams of graph-structured data
- STING status and immediate plans
- Community detection and monitoring

Observations

Exascale Data Analysis

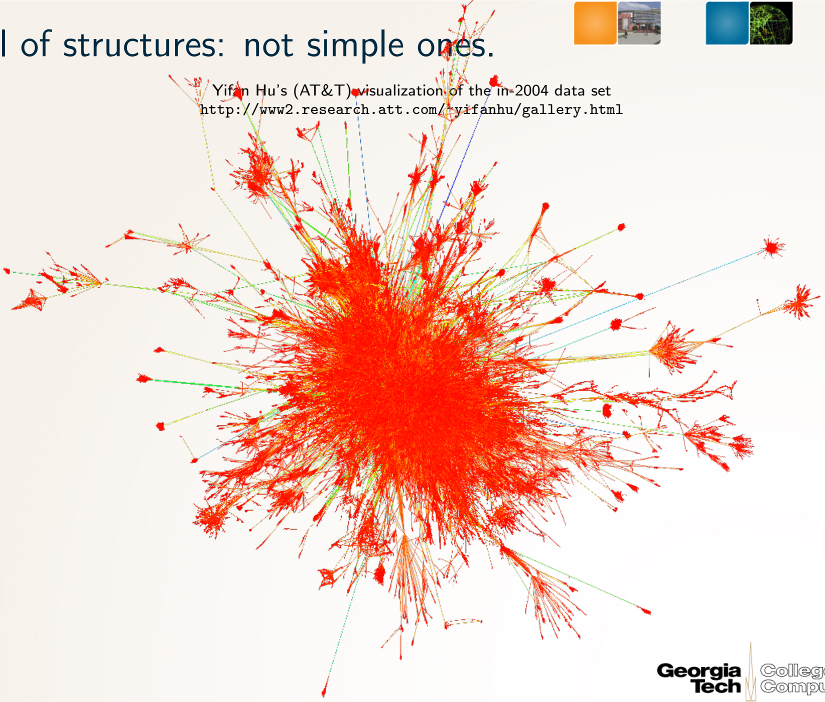


Health care	Finding outbreaks, population epidemiology
Social networks	Advertising, searching, grouping
Intelligence	Decisions at scale, regulating algorithms
Systems biology	Understanding interactions, drug design
Power grid	Disruptions, conservation
Simulation	Discrete events, cracking meshes

The data is full of semantically rich relationships.
Graphs! Graphs! Graphs!

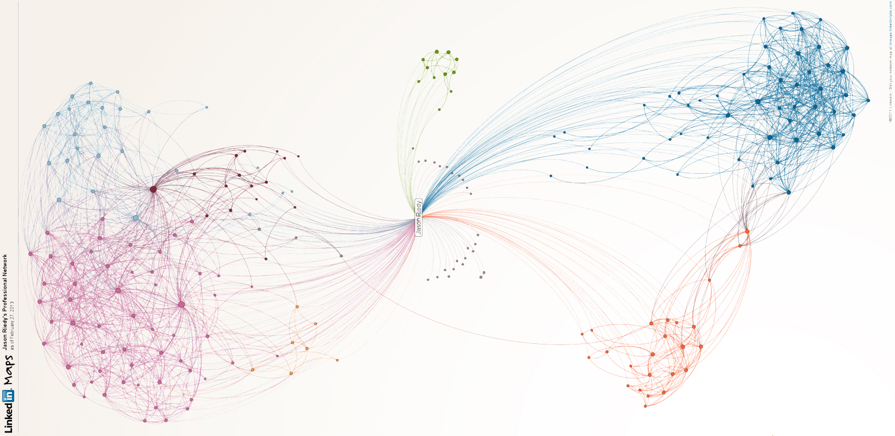
Full of structures: not simple ones.

Yifan Hu's (AT&T) visualization of the in-2004 data set
http://www2.research.att.com/~yifanhu/gallery.html



Full of structures: not well-defined ones.

LinkedIn Labs Map of my network
`http://inmaps.linkedinlabs.com`



No shortage of data...

Existing (some out-of-date) data volumes

- **NYSE**: 1.5 TB generated daily into a maintained 8 PB archive
- **Google**: "Several dozen" 1PB data sets (CACM, Jan 2010)
- **LHC**: 15 PB per year (avg. 21 TB daily) `http://public.web.cern.ch/public/en/lhc/Computing-en.html`
- **Wal-Mart**: 536 TB, 1B entries daily (2006)
- **EBay**: 2 PB, traditional DB, and 6.5PB streaming, 17 trillion records, 1.5B records/day, each web click is 50-150 details. `http://www.dbms2.com/2009/04/30/ebays-two-enormous-data-warehouses/`
- **Faceboot**: > 1B monthly users...

- All data is *rich* and *semantic* (**graphs!**) and changing.
- Base data rates include items and not *relationships*.

General approaches

- High-performance *static graph analysis*
  - Develop techniques that apply to unchanging massive graphs.
  - Provides useful after-the-fact information, starting points.
  - Serves many existing applications well: market research, much bioinformatics, ...
  - Needs to be $O(|E|)$.
- High-performance streaming graph analysis
  - Focus on the dynamic changes within massive graphs.
  - Find trends or new information as they appear.
  - Serves upcoming applications: fault or threat detection, trend analysis, ...
  - Can be $O(|\Delta E|)$? $O(\mathrm{Vol}(\Delta V))$? Less data $\Rightarrow$ faster, cheaper.

Both very important to different areas.
Remaining focus is on streaming.
Note: Not CS theory streaming, but analysis of streaming data.

Why analyze data streams?

Data volumes

NYSE	1.5TB daily
LHC	41TB daily
Facebook	Who knows?

Data transfer

- 1 Gb Ethernet: 8.7TB daily at 100%, 5-6TB daily realistic
- Multi-TB storage on 10GE: 300TB daily read, 90TB daily write
- CPU $\leftrightarrow$ Memory: QPI,HT: 2PB/day@100%

Data growth

- Facebook: $> 2\times$/yr
- Twitter: $> 10\times$/yr
- Growing sources: Bioinformatics, μsensors, security

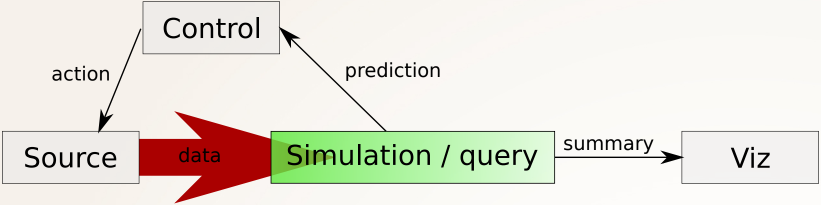
Speed growth

- Ethernet/IB/*etc.*: $4\times$ in next 2 years. Maybe.
- Flash storage, direct: $10\times$ write, $4\times$ read. Relatively huge cost.

STING: the How-To Part

- STING: Spatio-Temporal Interaction Networks and Graphs
- Shared-memory framework (in progress) for
  - inserting and removing typed edges and vertices in batches,
  - running analysis kernels on pre- and post-combinations of
    - the batch of changes,
    - the graph,
    - other kernels' results,
    - ...
- STINGER data structure maintains the graph.
- Free software available through `http://www.cc.gatech.edu/stinger/`, supports OpenMP and Cray XMT.
- Generally, *STING* is the framework and *STINGER* is the data structure, but we often say STINGER.

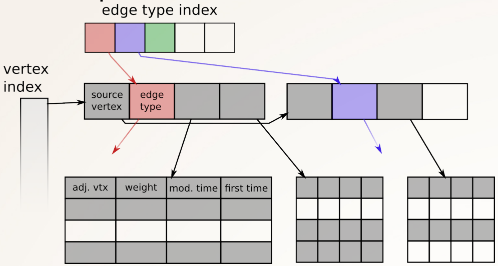
STING's focus



- STING manages queries against changing graph data.
  - Visualization and control often are application specific.
- Ideal: Maintain many persistent graph analysis kernels.
  - Keep one current snapshot of the graph resident.
  - Let kernels maintain smaller histories.
  - Also (a harder goal), coordinate the kernels' cooperation.
- Gather data into a typed graph structure, STINGER.

STINGER

STING Extensible Representation:



- Rule #1: No explicit locking.
  - Rely on atomic operations, or
  - tolerate data (but not data structure) inconsistency.
- Rule #2: In memory...
- Massive graph: Scattered updates, scattered reads rarely conflict. Be optimistic whenever possible.
- Use time stamps for some view of time.

STING status and immediate plans

Current distribution: `http://www.cc.gatech.edu/stinger/`

- Oriented towards research and demonstration.
- Included kernels:
  - insertion and removal demo / benchmark [1],
  - updating clustering coefficients [2, 3],
  - updating connected components [4, 3],
  - community re-agglomeration,
  - static benchmarks: BFS, components.
- Available:
  - incremental betweenness centrality* [5].
- Uses:
  - Multiple internal projects.
  - External collaborators at Intel, CMU, SAIC, Nobilis, and others.

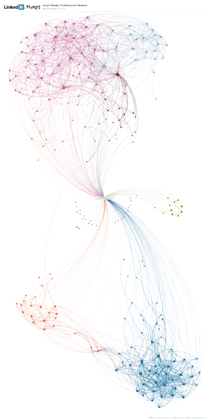
Development: `rad` branch

- Framework supporting communicating, run-time work-flow.

Community detection

What do *we* mean?

- Partition a graph's vertices into disjoint communities / clusters.
- A community locally maximizes some metric...
- No single accepted hard definition.
- Try to capture that vertices are *more similar* within one community than between communities.

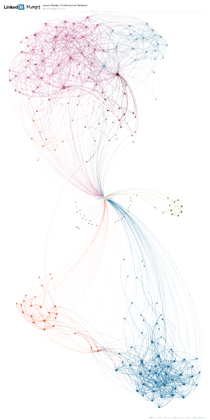


Jason's network via LinkedIn Labs

Community detection

Assumptions

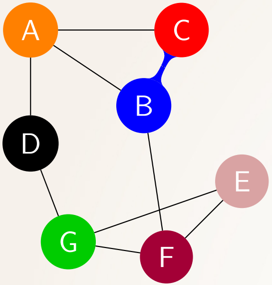
- Disjoint partitioning of vertices.
- There is no one unique answer.
  - Many metrics are NP-complete to optimize or just plain ill-defined.
  - Graph is lossy representation.
- Want an adaptable detection method.



Jason's network via LinkedIn Labs

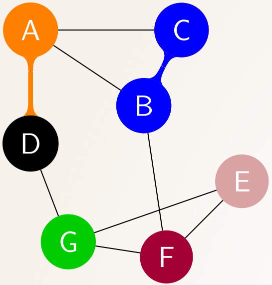
Georgia Tech College of Computing

Sequential agglomerative method



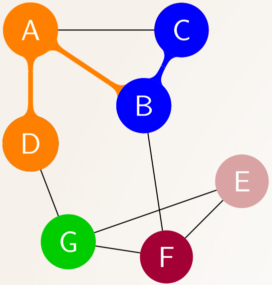
- A common method (*e.g.* Clauset, *et al.* [6]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity (Wakita & Tsurumi [7]).

Sequential agglomerative method



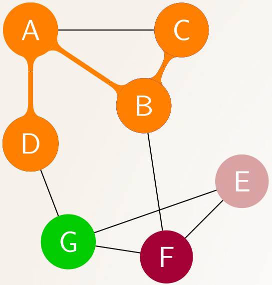
- A common method (*e.g.* Clauset, *et al.* [6]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity (Wakita & Tsurumi [7]).

Sequential agglomerative method



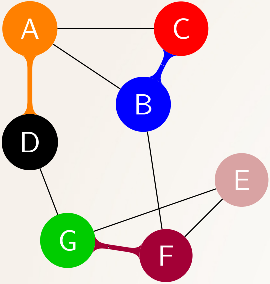
- A common method (*e.g.* Clauset, *et al.* [6]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity (Wakita & Tsurumi [7]).

Sequential agglomerative method



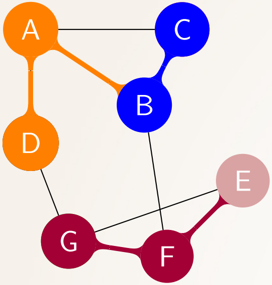
- A common method (*e.g.* Clauset, *et al.* [6]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity (Wakita & Tsurumi [7]).

Parallel agglomerative method



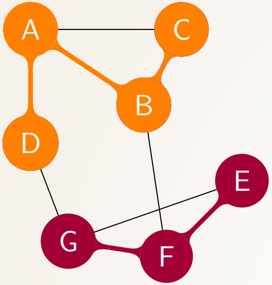
- We use a matching to avoid the queue. [8, 9, 10]
- Compute a heavy weight, large matching.
  - Simple greedy algorithm.
  - Maximal matching.
  - Within factor of 2 in weight.
- Merge all matched communities.
- Maintains some balance.
- *Produces different results.* Fast.
- Agnostic to weighting, matching...
  - Can maximize modularity, minimize conductance.
- Think AMG, ParMetis, ... Effectively $\tilde{O}(|E|)$.

Parallel agglomerative method



- We use a matching to avoid the queue. [8, 9, 10]
- Compute a heavy weight, large matching.
  - Simple greedy algorithm.
  - Maximal matching.
  - Within factor of 2 in weight.
- Merge all matched communities.
- Maintains some balance.
- *Produces different results.* Fast.
- Agnostic to weighting, matching...
  - Can maximize modularity, minimize conductance.
- Think AMG, ParMetis, ... Effectively $\tilde{O}(|E|)$.

Parallel agglomerative method



- We use a matching to avoid the queue. [8, 9, 10]
- Compute a heavy weight, large matching.
  - Simple greedy algorithm.
  - Maximal matching.
  - Within factor of 2 in weight.
- Merge all matched communities.
- Maintains some balance.
- *Produces different results.* Fast.
- Agnostic to weighting, matching...
  - Can maximize modularity, minimize conductance.
- Think AMG, ParMetis, ... Effectively $\tilde{O}(|E|)$.

Adapting for streaming data

Static implementation notes

- Uses a simple binned edge list.
- Only stores undirected edges, not pairs of directed edges.
- Very un-STINGER.
- Example of adapting an existing fast code into STING.

Simplest thing that could work.

- Parallel agglomeration produces a contracted *community graph*.
- Community graph vertex: set of original graph vertices
- *Batch* of edge insertions and removals touches a subset of original graph vertices.
- *Extract* using STINGER, then *re-start* agglomeration.
- Consistent with *some* contraction ordering starting from scratch, possibly not locally maximal.

Extracting vertices, edges

Before changing STINGER graph

1. Collect unique vertices from edge change batch.
   - (common operation across kernels)
2. Extract affected vertices into new communities.
   - Except the last one... Use community volume to check.

Given *changed* STINGER graph

1. Append corrective edges to community graph edge list.
   - Append edge with negative weight to cancel old edge between old communities, positive weight to link new communities.
   - (Edge removals handled similarly.)

- Operations $\propto$ to volume of affected vertices.
- Re-agglomeration $\propto$ edges in community graph plus batch size.

Performance: Setup

Test cases

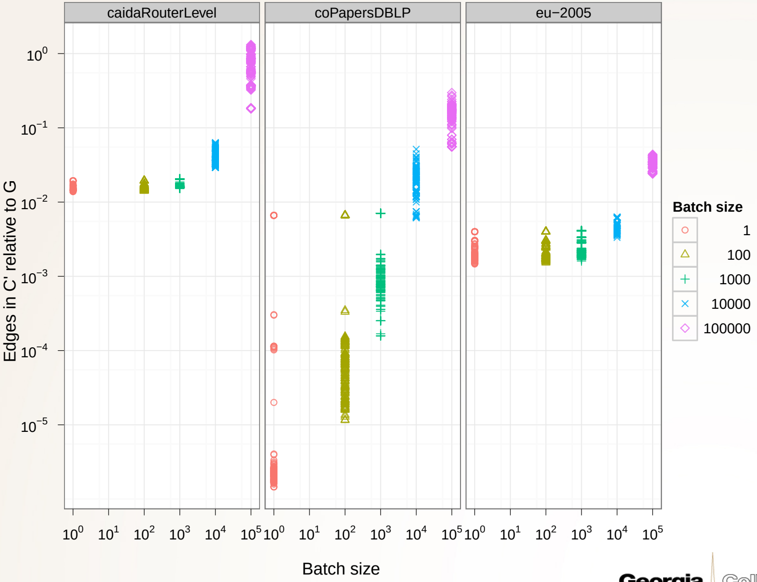
- Three graphs from DIMACS10, initial communities:

Name	$\lvert V \rvert$	$\lvert E \rvert$	$\lvert C \rvert$	$\lvert E_C \rvert$
caidaRouterLevel	192 244	1 218 132	18 343	30 776
coPapersDBLP	540 486	30 866 466	1 401	205 856
eu-2005	862 664	16 138 468	55 624	194 971

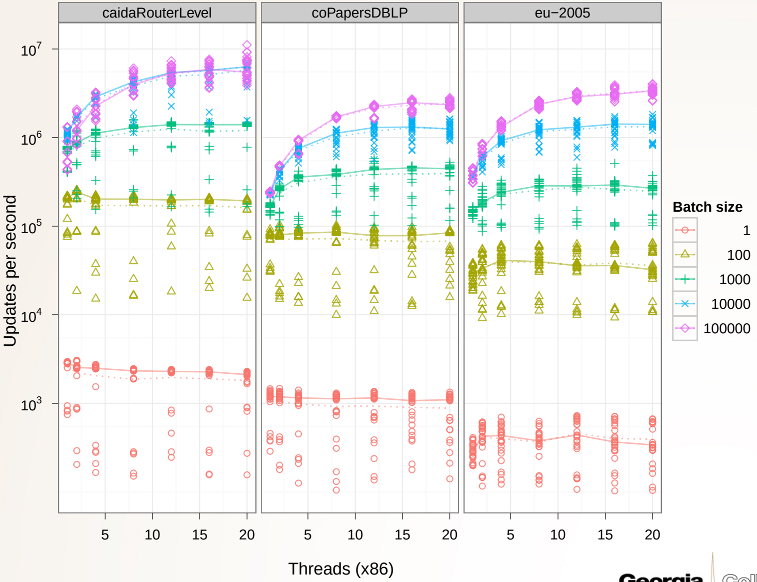
- Edge action generation: 15/16 insertion, 1/16 removal
  - Insertion: Between two communities ($\propto$ density), select endpoints with prob. inversely proportional to degree.
  - Removal: Randomly sample from existing edges inside communities, then from inserted edges if exhausted.
- Only for execution performance, not quality. Using a real stream for quality is in progress.

Platform: Quad 6-core, 2.0GHz Intel Westmere-based system with 1 TiB RAM (courtesy or Oracle).

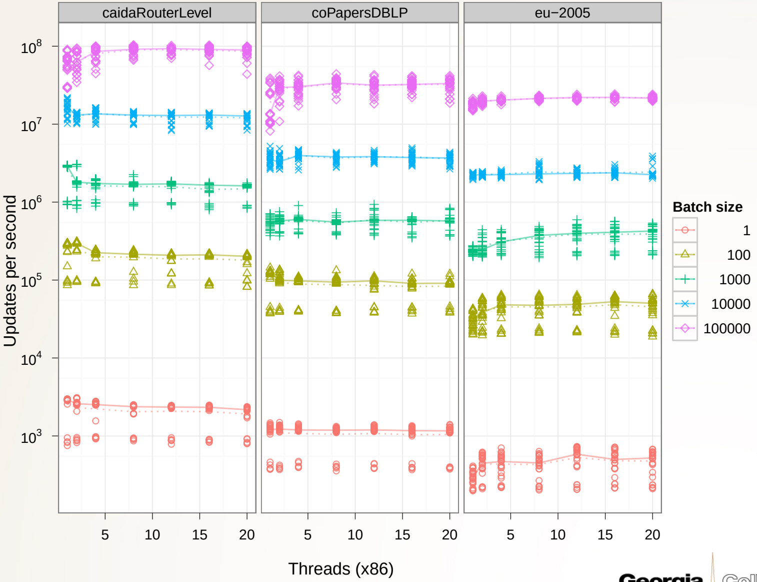
Community graph expansion



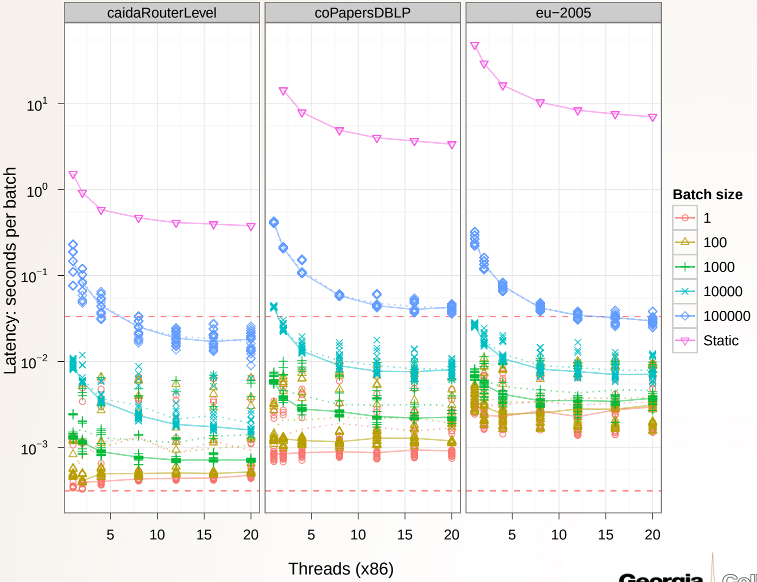
Updates per second, w/STINGER



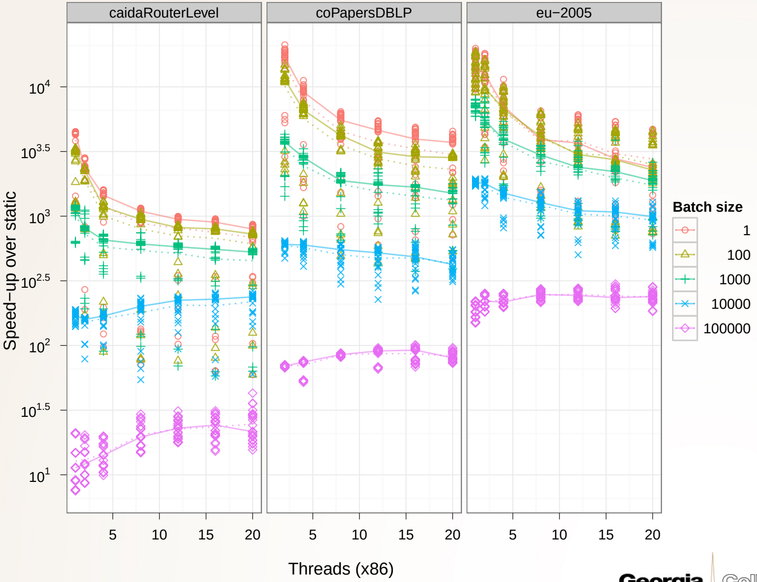
Updates per second, only re-agglom.



Latency, w/STINGER



Speed-up over static



Observations

- Straight-forward to adapt some surprising graph algorithms to streaming data.
- STING provides a useful platform for quick development.
  - Good base-line performance on general (OpenMP) and specialized (XMT) architectures.
  - Tunable latency v. raw throughput (updates/sec).
  - Developing a work-flow and blackboard model.
  - External users finding issues, *e.g.* lower performance on high-degree vertices.
    - Good and bad.

http://www.cc.gatech.edu/stinger

Georgia Tech College of Computing

Acknowledgment of support

Bibliography I

D. Ediger, R. McColl, J. Riedy, and D. A. Bader, "STINGER: High performance data structure for streaming graphs," in *The IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, Sep. 2012, best paper award.

D. Ediger, E. J. Riedy, D. A. Bader, and H. Meyerhenke, "Tracking structure of streaming social networks," in *5th Workshop on Multithreaded Architectures and Applications (MTAAP)*, May 2011.

J. Riedy, H. Meyerhenke, D. A. Bader, D. Ediger, and T. G. Mattson, "Analysis of streaming social networks and graphs on multicore architectures," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Kyoto, Japan, Mar. 2012. [Online]. Available: http://www.slideshare.net/jasonriedy/icassp-2012-analysis-of-streaming-social-networks-and-graphs-on-mult

Georgia Tech College of Computing

Bibliography II

- D. Ediger, K. Jiang, E. J. Riedy, and D. A. Bader, "Massive streaming data analytics: A case study with clustering coefficients," in *4th Workshop on Multithreaded Architectures and Applications (MTAAP)*, Atlanta, GA, Apr. 2010. [Online]. Available: http://www.cc.gatech.edu/~bader/papers/StreamingCC.html
- O. Green, R. McColl, and D. A. Bader, "A fast algorithm for incremental betweenness centrality," in *ASE/IEEE International Conference on Social Computing (SocialCom)*, Amsterdam, The Netherlands, Sep. 2012.
- A. Clauset, M. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, no. 6, p. 66111, 2004.
- K. Wakita and T. Tsurumi, "Finding community structure in mega-scale social networks," *CoRR*, vol. abs/cs/0702048, 2007.

Bibliography III

E. J. Riedy, H. Meyerhenke, D. Ediger, and D. A. Bader, "Parallel community detection for massive graphs," in *9th International Conference on Parallel Processing and Applied Mathematics (PPAM11)*. Springer, Sep. 2011.

E. J. Riedy, D. A. Bader, and H. Meyerhenke, "Scalable multi-threaded community detection in social networks," in *6th Workshop on Multithreaded Architectures and Applications (MTAAP)*, May 2012. [Online]. Available: http://www.slideshare.net/jasonriedy/mtaap12-scalable-community-detection

Bibliography IV

E. J. Riedy, H. Meyerhenke, D. Ediger, and D. A. Bader, "Parallel community detection for massive graphs," in *10th DIMACS Implementation Challenge Workshop - Graph Partitioning and Graph Clustering*. Atlanta, Georgia: (workshop paper), Feb. 2012, won first place in the Mix Challenge and Mix Pareto Challenge. [Online]. Available: http://www.cc.gatech.edu/dimacs10/papers/[15]-dimacs10-community-detection.pdf