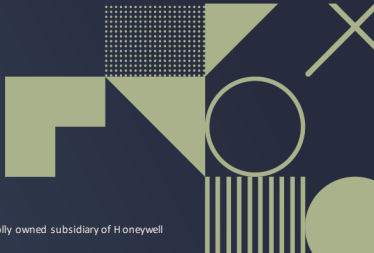
ElGA: Elastic and Scalable Dynamic Graph Analysis

Kasimir Gabert[1,2], Kaan Sancak[1], M. Yusuf Özkaya[1], Ali Pınar[2], Ümit V. Çatalyürek[1,3]

[1]Georgia Institute of Technology

[2]Sandia National Laboratories

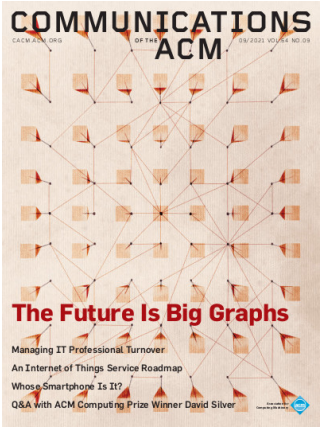
[3]Amazon Web Services. This publication is not associated with Amazon



SAND 2021-12989 C. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Large Graphs Are Everywhere—and Continuously Changing

- Graphs are a flexible data structure for data science
- Graphs are not only large, but continuously changing in bursts
  - Large: number of edges can be billions to trillions
  - Rapidly changing: e.g., Twitter averages 6k changes-per-second
  - Bursty: e.g., Twitter bursted to 140k c/s; GitHub traffic was 127m c/s for 20 mins



Cover of Communications of the ACM, September 2021.

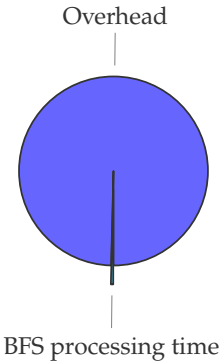
We Need Scalable, Dynamic, Elastic Graph Systems

distributed: graphs exceed shared memory

dynamic: graphs continuously change

elastic: system itself must change

- Is it possible for a high-performance system to be elastic and support dynamic graphs?



Static systems don't work: Graphalytics 2018 winner's BFS time (0.5 sec) and overhead time (141 sec)

Goals for Such a System

On a P processor system, for a graph with n vertices and m edges:

goal 1: Supports large graphs ($m \geq 100$ B)

goal 2: Memory per machine is in expectation $O((n + m)/P + P)$

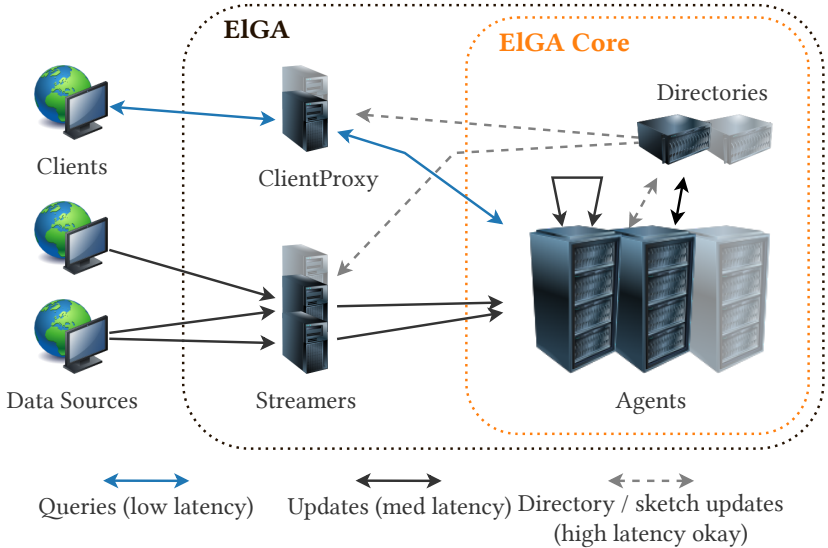
goal 3: System partition lookups depend on P, not input graph

goal 4: Dynamic graph support (edge and vertex insertion and deletion)

goal 5: System can elastically scale (adding, removing processors)

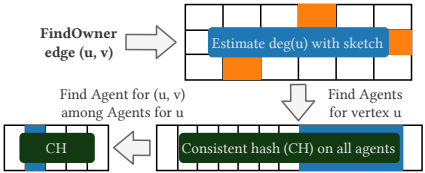
We show these goals are all achievable, without losing performance

Components of ElGA



EIGA's Approach

- Shared-nothing architecture
- Edges partitioned with consistent hashing
- A sketch keeps track of degrees
  - High-degree vertices can be partitioned
- Vertex-centric computation is supported
- Bulk-synchronous parallel and finer-grained vertex activations are supported



Note: no dependence on n or m

Example Algorithm: Weakly Connected Components

"Think like a vertex": each vertex can send messages and compute from neighbors

Endpoints become active on inserted edges

```
1  void init() { id = v; } // Initialize the local vertex state
2  void run() { // Process as a vertex
3      auto old_id = id;
4      for (auto r : replica_msgs) id = min(id, r);
5      for (auto n : neighbor_msgs) id = min(id, n);
6      if (old_id != id) broadcast(id);
7      state = INACTIVE;
8  }
9  void set_active(msg) { // Runs when receiving message
10     if (msg < id) state = ACTIVE;
11 }
```

What About Dynamic Algorithms?

- The vertex-centric approach is suitable for many static problems
- What about operating on a batch of edge changes?
- A locally persistent algorithm:
  - stores memory only on vertices (or edges)
  - only activates vertices that are neighbors of active vertices
  - is considered bounded if all activated vertices are a polynomial function of vertices with ultimately changed outputs

Locally Persistent Algorithms Are Similar to Vertex-Centric

- Both only have local storage, active vertices, and messages to neighbors
- Difference in synchronization and message recipients

ElGA has three message models:

BSP	a full bulk-synchronous parallel model, where every vertex sends a message to all neighbors at each iteration
LBSP	"light" BSP, only active vertices send messages
full	a flexible model, vertices send messages to specific neighbors

Algorithms choose the model to use, with performance tradeoffs in each level.

Implementation Details

- C++17
- Uses ZeroMQ for messaging
- Uses Abseil for flat hash maps

- Available at `https://github.com/GT-TDAlab/ElGA`

- Implemented static algorithms: BFS, WCC, PageRank, LPA, k-core
- Well suited for locally persistent dynamic algorithms

Experimental Setup

- 65 servers, dual-socket Intel Broadwell, 16 cores each, 512 GB RAM
- Connected with 100 Gbps Ethernet to an Arista 7500E

Baselines:

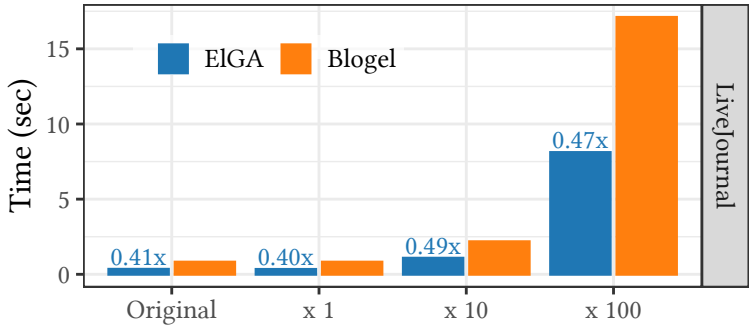
Blogel[1]	static, chosen for fastest end-to-end times
GraphX[2]	dynamic, chosen as a platform for other implementations

Controlled for identical algorithms and parameters

[1] `https://www.cse.cuhk.edu.hk/blogel/`, Yan et al., 2014.
[2] `https://spark.apache.org/graphx/`, Gonzalez et al., 2014.

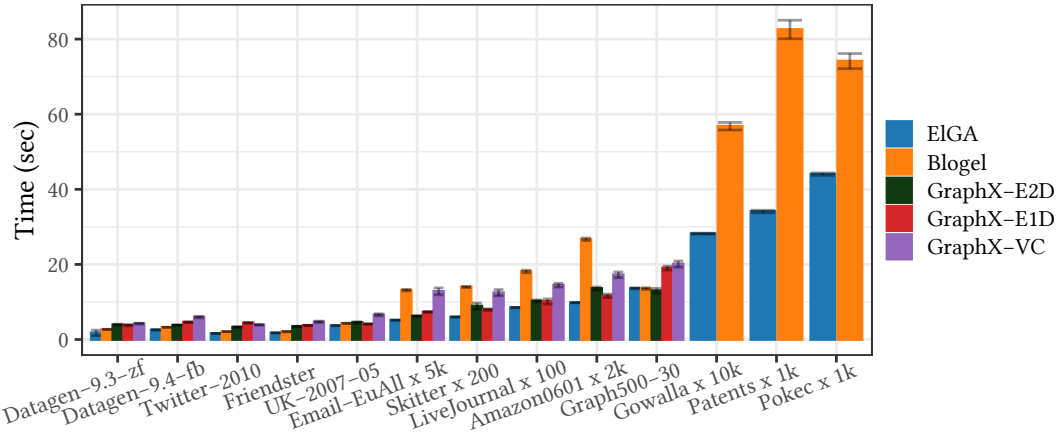
Datasets and Scaling Them with A-BTER

- Benchmark datasets from LAW, SNAP, and LDBC
- Both original size and A-BTER scaled graphs, 25 GB to 2.3 TB



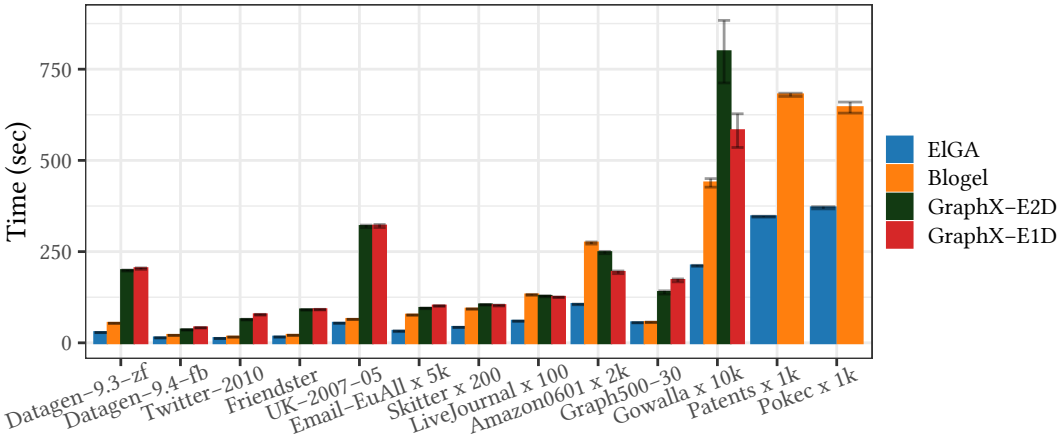
- Relative performance ratio remains similar as graph is scaled with A-BTER

Faster PageRank Iterations Than Static Systems



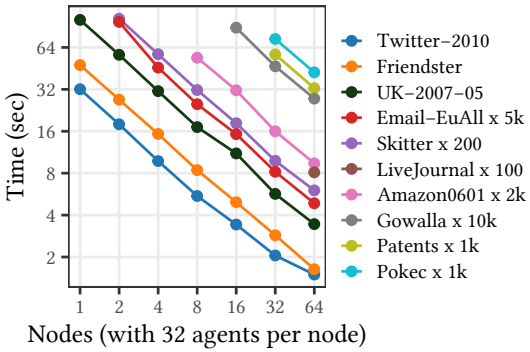
- In all cases, ElGA is fastest
- Baselines have expensive partitioning phases (not shown); ElGA does not

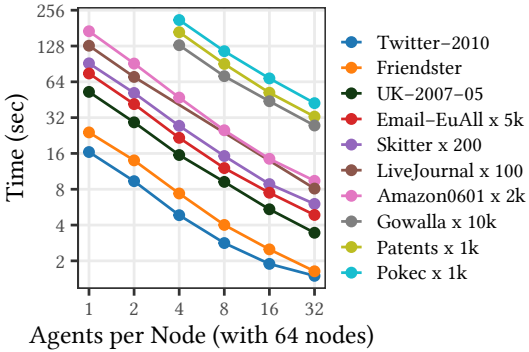
Faster Connectivity Computation Than Static Systems

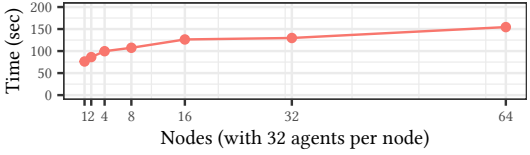


- Similarly, ElGA is fastest for weakly connected components
- May be due to asynchronous messages in the shared-nothing architecture

Scales Well With Both Strong and Weak Scaling

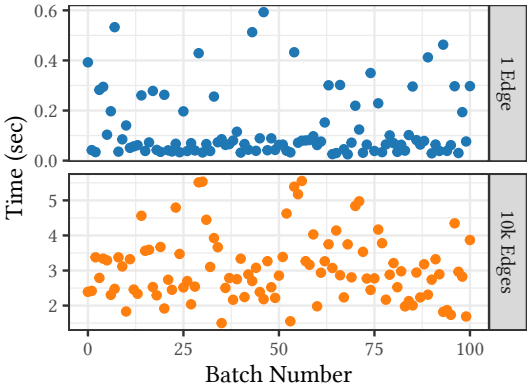




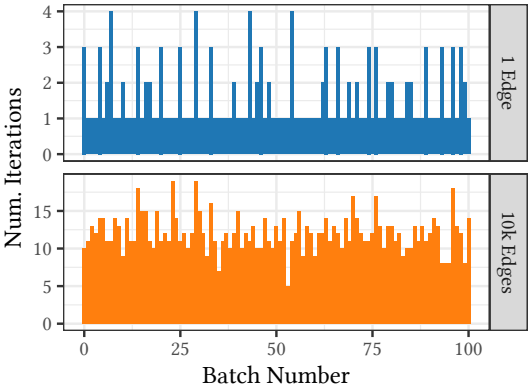


- Close to linear strong scaling
- Close to horizontal weak scaling

Dynamic Algorithms Can Run End-to-End Quickly



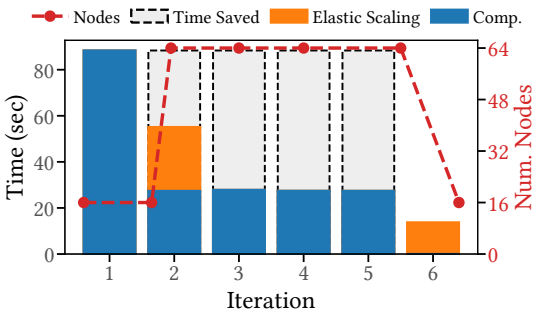
Runtime for maintaining connectivity on Twitter-2010



Iterations to maintain connectivity

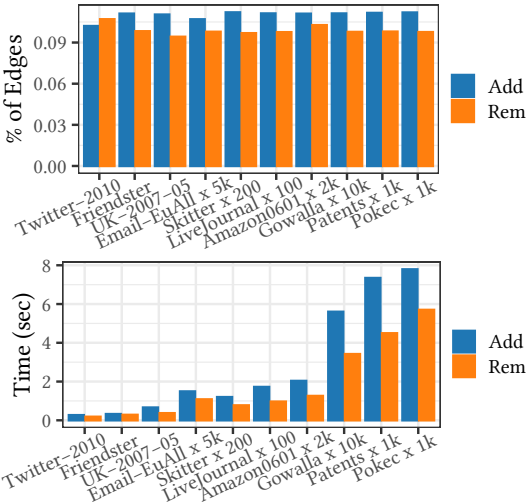
- GraphX takes 49 seconds to start, 6 seconds per iteration (not shown)
- EIGA can maintain components in seconds, without large overheads

Elasticity Is Quick and Has a Local Impact



PageRank on Gowalla (28 B edges)

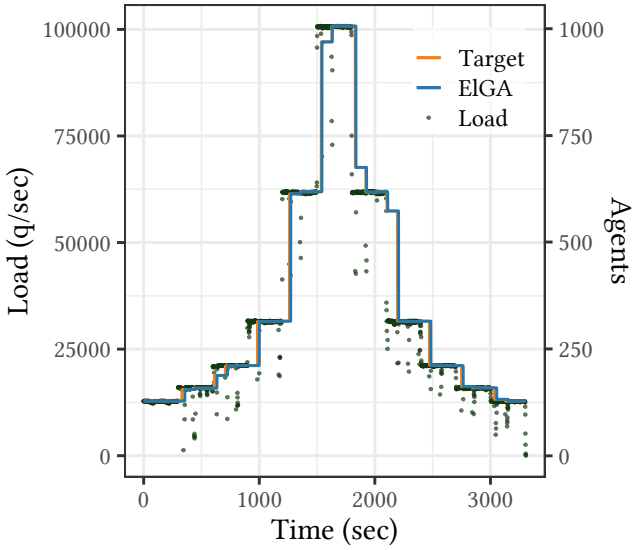
- Add/del resources at runtime
- Only impacted edges are re-partitioned



Transfer costs from adding and removing an agent

Autoscaling Enables Optimizations for Specific Objectives

- Autoscaling takes an operator out of the loop
- We implemented an exponential moving average autoscaler using the client query rate
- ElGA matches the autoscaler's target quickly



Conclusion

- We highlight the need for both elasticity and dynamic graph support
- We proposed ElGA
  - a high-performance graph system that is elastic and supports dynamic graphs
  - with an easy to use vertex-centric programming model, avoiding specialized, elasticity-aware algorithms
  - supporting locally persistent dynamic graph algorithms
- We experimentally evaluate ElGA on a range of graphs and show superior performance to static baselines

THANK YOU

Any comments or questions? Please contact tda@cc.gatech.edu!