DynoGraph
Benchmarking Dynamic Graph Analytics

Eric Hein <ehein6@gatech.edu>
Tom Conte <tom@conte.us>

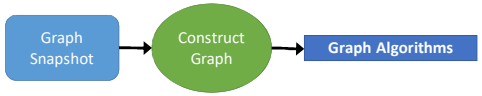
Georgia Institute of Technology

Abstract

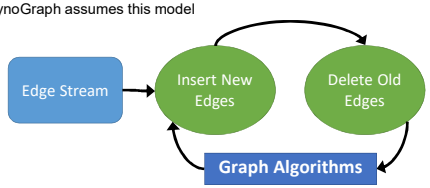
Large scale graph processing is the key to understanding complex relationships, ranging from the interaction of people on social media to the flow of data through a corporate computer network. Graph analytics present unique challenges for HPC system designers since they lack data locality and are difficult to partition into equally-sized units of work. In addressing these challenges, many researchers have opted to work with static graphs instead of the more difficult case of dynamic graphs that change rapidly during the analysis. Dynamic graphs require an entirely different memory layout, leading to degraded performance as algorithms traverse a fragmented, unsorted graph data structure. DynoGraph provides a standard for benchmarking dynamic graph analytics engines, bringing needed focus to this important class of applications.

Motivation

- **Static Graph Workflow**
  - Load the graph all at once from disk
  - Edges remain sorted and contiguous in memory
  - Most graph benchmarks assume this workflow



- **Dynamic Graph Workflow**
  - Continuously stream in edge updates
  - Edges become unsorted and fragmented in memory over time
  - DynoGraph assumes this model

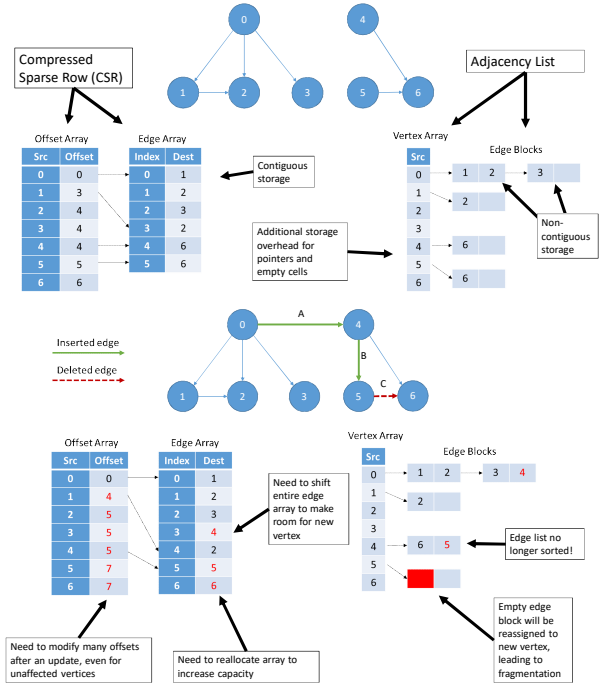


DynoGraph

DynoGraph is a benchmark suite for dynamic graph analytics engines. It aims to accurately capture the unique performance characteristics of dynamic graphs by focusing not only on algorithms, but also on the performance of edge insertions and deletions and their effect on the efficiency of the graph layout. DynoGraph is designed to aid computer architects in satisfying the needs of dynamic graph applications when designing processing-near-memory accelerators for emerging high-bandwidth memory systems.

Graph Data Structures

- The way a graph is stored in memory affects the performance of graph analytics. The Compressed-Sparse-Row (CSR) format stores graphs more efficiently, but an adjacency list is easier to update.



Graph Inputs

- Netflow data from SCinet 2015 (sc15-10K)
  - In this graph, each vertex represents a SCinet IP address. An edge represents that data was transferred between those two hosts. Edges are weighted with the number of bytes transferred over the analysis time window considered. This dataset is representative of the types of graphs generated when analyzing real networks for cyber security threats where the application of PageRank, betweenness centrality, and community detection are used to find botnets, emergent graph behavior, and potential distributed denial of service (DDoS) attack targets
- Passive DNS (dns-100K)
  - This dataset is an anonymized graph of real DNS data collected over the entire campus network of a large university. Edges in this graph represent domain name lookups for a given host as they occur in time, and track the resolution of the name requests as they traverses the hierarchy of name servers. Real-time applications of this data include using centrality and community detection to locate hackers and bot-net control networks, which maliciously manipulate DNS records to accomplish their goals.
- Twitter data from the 2014 World Cup (worldcup-100K)
  - Social media remains one of the most prevalent applications of dynamic graph analysis. This data set represents the Twitter graph of mentions between Twitter users that occurred over a three week period during the 2014 World Cup. The Twitter graph was collected targeting hash tags specific to the World Cup as well as hash tags specific to the Twitter campaigns of the primary sponsors.
- All datasets were anonymized prior to analysis

Results

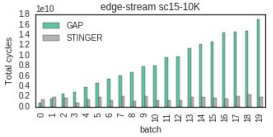
The following experiments compare GAP, a graph benchmark that uses CSR, against STINGER, a dynamic graph engine that implements a parallel adjacency list.

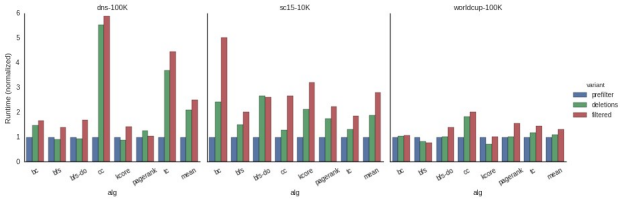
Due to the use of a less efficient dynamic data structure, STINGER algorithms take much longer to run than their counterparts in GAP.

Slowdown of STINGER vs GAP implementations of DynoGraph

algorithm	graph		
	dns-100K	sc15-10K	worldcup-100K
bc	31.53x	296.70x	74.88x
bfs	0.35x	18.61x	3.85x
cc	3.46x	14.28x	14.35x
pagerank	45.00x	130.02x	27.48x
tc	19359.52x	495.29x	1759.82x

However, the static graph representation can't keep up with the rate of incoming edge updates. Since GAP has to reconstruct the graph for each batch of updates, its runtime is proportional to the size of the graph, while STINGER's runtime is proportional to the size of the batch.





The performance difference between processing a freshly loaded graph and a fragmented one can be significant. In this experiment, the graph algorithms process a subset of the graph based on a time window. The "holes" left in the graph by deletions degrade performance, but not as much as filtering on-the-fly.

Variant	Presorted?	Filtered?	Filtering method
filtered	No	Yes	Runtime timestamp filter
deletions	No	Yes	Delete old edges
prefilter	Yes	Yes	Offline timestamp filter
baseline	No	No	None

Conclusions

- Dynamic graph analytics represents an entirely different workflow from static graph processing, and deserves separate attention.
- The in-memory layout of a dynamic graph degrades over time as edges are inserted and deleted. It is important to model this effect when measuring performance.
- DynoGraph provides a framework for accurately benchmarking dynamic graph processing on different engines and architectures.