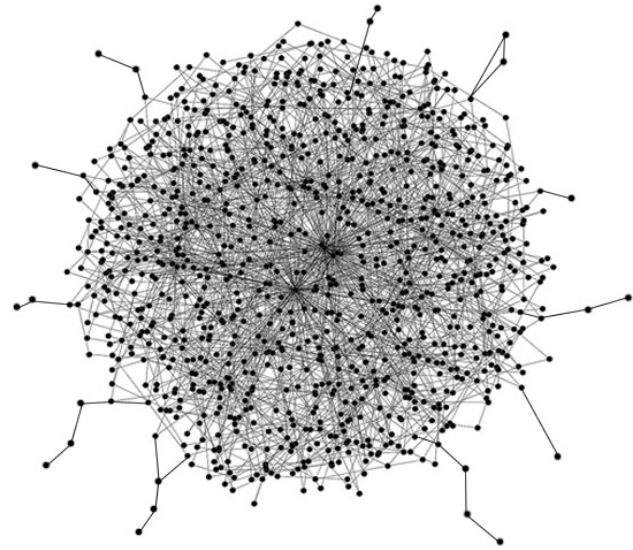
GraphIn

ENDRIAS KAHSSAY

Motivation

- Real world graphs are huge.
- -Facebook Graph has over a billion vertices.



- Real world graph topologies change often.
- -Facebook’s graph changes 86000 objects/ second
- -2.5 emails are sent /sec

- The update usually affect small parts of the graph.

Current Approach

- Graphs are popular -- many graph processing frameworks like GraphLab, PowerGraph, Graphchi
- Most frameworks deal with graph updates in batches.
- For each batch, they do a static computation on the entire graph after applying the batch.
- Simple for the programmer, but very slow for large graphs.
- -Often only a small part of the computation changes.

GraphIn

- A high-performance incremental graph processing framework
  - built on top of GraphMat to process time-varying evolving graphs.

- Incrementally process a continuous stream of updates (i.e., edge/vertex insertions and/or deletions) as a sequence of batches

GraphIn

- Based on a new framework called Incremental-Gather-ApplyScatter (or I-GAS)
- Allows simple expression of incremental versions of popular algorithms
- User has to implement 6 serial functions.
- -meta computation(), build inconsistency list(), check property(), activate frontier(), update - inconsistency list() and merge state().
- The library will generate highly optimized parallel code.

GAS

GraphIn is based on the popular Gather-Apply-Scatter.

Gather phase: incoming messages are processed and combined (reduced) into one message.

Apply phase: vertices use the combined message to update their state.

Scatter phase: send messages to their neighbors.

Repeats until no more active vertices.

GraphIn

GraphIn uses compressed matrix format to store static version of the graph

Allowed fast static computation over the graph using GraphMat.

Uses edge list to store incremental updates.

Merges update list and static graph when needed.

Phase I

Static Graph Computation and Meta Computation to be used for later stage.

This computation follows the GAS model

→ any framework supporting GAS can be used for static version

-> GraphIn uses graphmat

Meta computation: properties of graphs such as vertex parents, degree

Phase II Inconsistency Graph Builder

- In an evolving graph, vertex states for many vertices remain the same over time.
- Identify inconsistent vertices: vertices where one or more properties are affected because of update
- Example: BFS, addition of edge (vi, vj) can potentially make vj and all vertices that are downstream from vj inconsistent.

Phase III: Property Check

- Framework calls check property() to decide static or dynamic computation.
- If there are a lot of affected vertices because of an update, incremental computation might be slower.
- Predefined graph properties such as vertex degree or user can define their own.
- In BFS, if a vertex will a low depth is removed, it might cause a lot of vertices to be inconsistent.

Phase IV: Incremental GAS Computation

- Ensures only the inconsistent part of the graph is recomputed.
- User implements the I-GAS program as well as the activate frontier() and update inconsistency list()

Algorithm 1. I-GAS computation loop per update batch

```
1: while(!inconsistency_list.isempty()):
2:     frontier = activate_frontier(G', inconsistency_list)
3:     IGAS(G')
4:     update_inconsistency_list(G', inconsistency_list, frontier)
```

Phase V: Merge Graph States

- Merges updates to vertex properties.
  → Example: new vertex depths calculated in incremental BFS
- Merge update batch with graph
  1. All-Merge: Both inserts and deletes are merged with static graph G.
  2. Partial-Merge: Either deletes or inserts are merged with G. The framework defers applying the rest of the updates to the original graphs.
  3. No-Merge: Neither inserts nor deletes from the update batch are merged with G. The framework defers applying both inserts and deletes.

	Graph Algorithms		
GraphIn APIs and Phases	Breadth First Search **(BFS)** [18]	Connected Components **(CC)** [8]	Clustering Coefficients **(CCof)** [6]
Type	All-merge	Delete-only-merge	No-merge
meta_computation() **(Phase I)**	Parent id and vertex degree	Vertex degree	Vertex degree
build_inconsistency_list() **(Phase II)**	**1.** Inconsistency list contains vertices with incorrect depth values with MIN_PRIORITY.	**1.** For each edge insertion add an edge in G' if the endpoints belong to different components [8, 15].	**1.** Inconsistency list contains endpoints of every edge inserted and/or deleted and their respective neighbors.
	2. G' = G	**2.** G' is also known as component graph	**2.** G' consists of inconsistent vertices and edge incident on them in G [6]

	BFS	Connected Components	Clustering Cofficents
check_property() **(Phase III)**	Check BFS depth property	Check disjoint component property	Check vertex degree property
activate_frontier() **(Phase IV)**	Activate inconsistent vertices with minimum depth value-Ramalingam and Reps [18]	Activate all the vertices in G'	Activate all the vertices in G'
update_inconsistency_list() **(Phase IV)**	Remove frontier vertices and add inconsistent successors to inconsistency list	Clear inconsistency list	Clear inconsistency list
merge_state() **(Phase V)**	**1.** Apply all insertions and deletions to G	**1.** Apply only deletions to G.	**1.** Applying insertions and deletions to G not required.
		2. Relabel components in G using G'	**2.** Update triangle counts and degree information in G using G'

Benchmark setup

- Dual-socket Intel node equipped with two Intel Xeon processors with 64 GB of DDR4 RAM.
- Updates are provided in batches of size ranging from 10,000 up to one million with 1 % of all updates being deletions.
- The endpoints of the edges used for batch updates are generated randomly.
- Algorithms Tested: Clustering Coefficient (CCof), Connected Components (CC) and BFS

Results:

- GraphIn achieves maximum speedups of 407×, 40× and 82× over static computation for (CCof), Connected Components (CC) and BFS.
- Note: BFS is done for 50 thousand batch size while 1 million batch size is used for the others.
- Average Speedup: 50x, 20x, 10x.
- GraphIn achieves up to 9.3 million updates/sec.
- Achieved from incremental computation for only inconsistent vertices

Effect of Graph Algorithm

Maximum speedup order:

1) no-merge algorithms such as clustering coefficients

2) partial-merge algorithm such as connectivity

3) and all-merge algorithms such as BFS

Dual Path execution

- BFS sensitive for depth threshold

  -> a small increment of 10 % in inconsistent vertices below the depth of 2 results in 29x decline in update rate

- Dual path execution achieve a maximum speedup of 60× for Facebook graph.

Comparison to Stinger

- Stinger is a competitive incremental framework.
- GraphIn had 6.6× speedup in throughput.
- STINGER, uses a single edge-list based data structures for both static and incremental graph processing.
- GraphIn's hybrid data structure enables faster computation both for static and dynamic.

Limitations:

- In the worst case, still has to a full static computation.
- Its not clear how GraphIn compares to parallel version of the static or incremental algorithms.
- Paper emphasis maximum speedup instead of average speedup.
- Unclear how merging works for differed updates. Also tests don’t factor cost of merging.

Future Work

Making GraphIn work on GPU

Evaluating this for a bigger cluster

Thanks for listening!