Graph Analytics using Vertica Relational Database

Alekh Jindal - Samuel Madden,
Malú Castellanos - Meichun Hsu

Introduction

- High demand for graph analytics

- Popularity of distributed graph computing systems
  - Vertex-centric systems: Pregel, Giraph, GraphLab

- Question:

Are traditional relational database systems not good enough for graph analytics?

Introduction

Limitations of distributed graph computing systems:

- Data is initially collected and stored in a relational database
- Graph processing is slow for very large graphs
  - Users have to choose a subgraph to run the algorithm
- Preparation might include operations that relational databases are optimized for.
  - Pre-processing or post-processing
- Some graph algorithms compute aggregates over a large neighbourhood
  - Hard to express in vertex-centric systems

Goal

- Show how vertex-centric graph processing can be translated, optimized and run on Vertica
    - SSSP, PageRank, Connected Components

- Compare Performance with two vertex-centric distributed systems for graph analysis (**Giraph** and **GraphLab**)

- Vertica → Enterprise column-store database management system
    - Supports parallel processing

Vertex-Centric Model

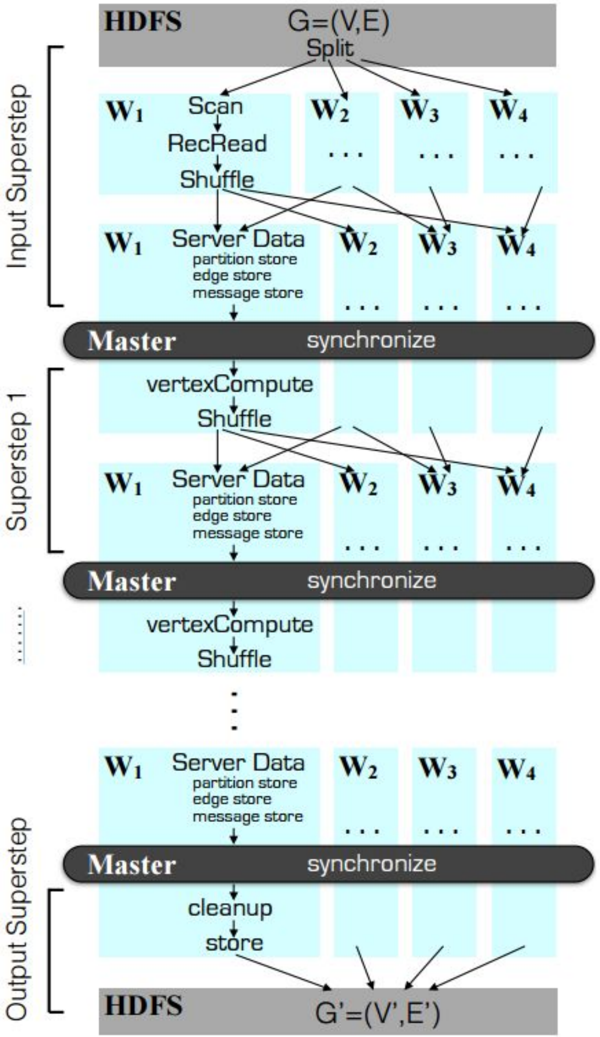
- The user provides a vertex.compute function (UDF):
  - The UDF will be executed at each node.
  - Will update the node’s state.
  - And communicate the changes to the neighbours.

Giraph Physical Plan

1. Input Superstep: Workers reading the data, building "Server Data stores"

2. Intermediate step: Run UDF, shuffle messages, wait for everyone, synchronize.

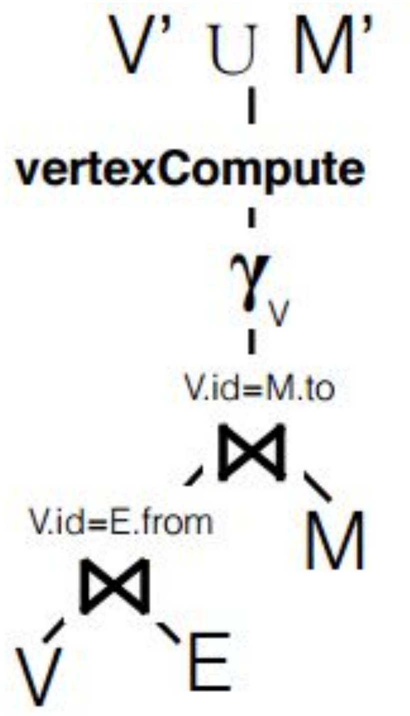
3. Output Superstep: Produce the output.



Giraph Logical Plan

Same query plan but in relational logic:

1. V join E
2. (V join E) join M: messages from previous superstep
3. Run UDF
4. Produce new state for vertex (V’) and messages for the next superstep (M’).

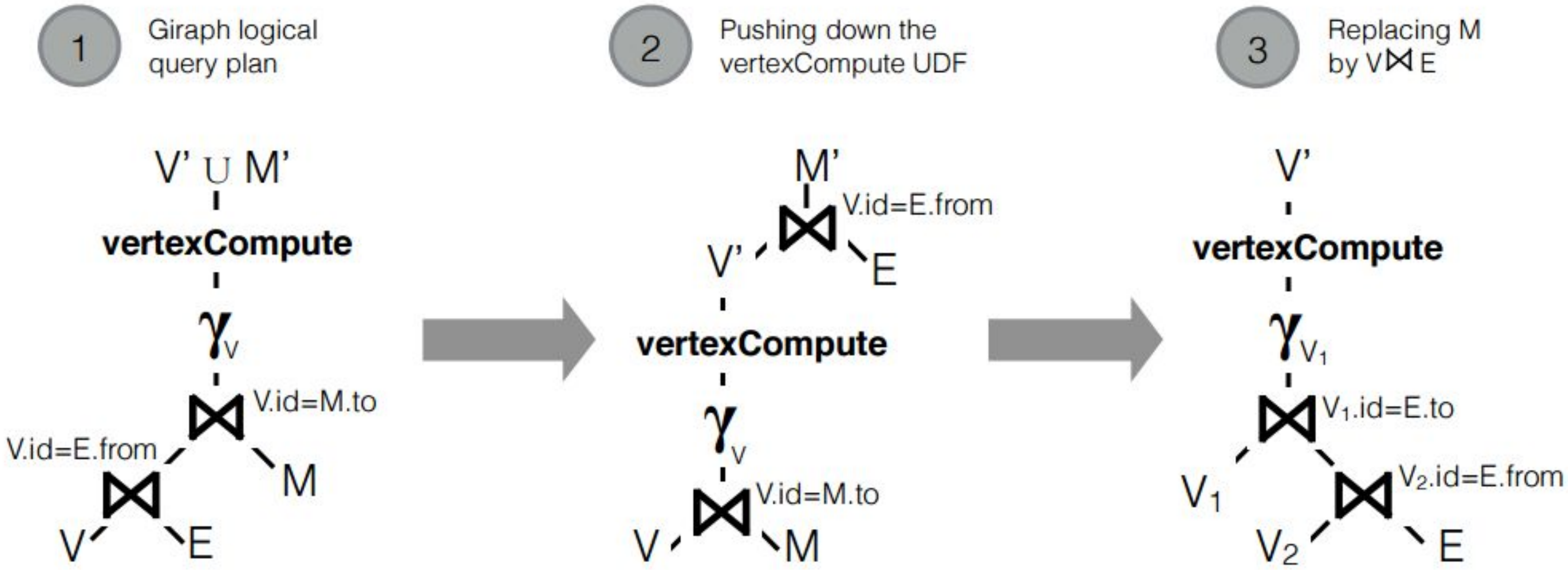


Overview

- Translation to SQL queries
- Query Optimization
- Query Execution
- Extending Vertica

Translation to SQL

1) Eliminate the message table

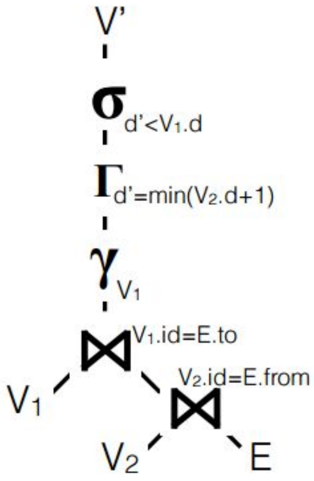


Translation to SQL

2) Translate vertex compute function

$$\text{SSSP} : \textit{vertexCompute} \longmapsto \sigma_{\text{d'}<V_1.d}(\Gamma_{\text{d'=min}(V_2.d+1)})$$

```
UPDATE vertex AS v SET v.d=v'.d
  FROM (
    SELECT v1.id, MIN(v2.d+1) AS d
      FROM vertex AS v1, edge AS e, vertex AS v2
      WHERE v2.id = e.from_node AND v1.id = e.to_node
      GROUP BY e.to_node, v1.d
      HAVING MIN(v2.d+1) < v1.d
    ) AS v'
    WHERE v.id=v'.id;
```



Logical plan

Query Optimizations

1) Update Vs. Replace

```
CREATE TABLE vertex_prime AS
  SELECT v.id, ISNULL(v'.d, v.d) AS d
    FROM vertex AS v LEFT JOIN (
      SELECT v1.id AS id, MIN(v2.d+1) AS d
        FROM vertex AS v1, edge AS e, vertex AS v2
        WHERE v2.id=e.from_node AND v1.id=e.to_node
        GROUP BY e.to_node, v1.d
        HAVING MIN(v2.d+1) < v1.d
    ) AS v'
    ON v.id = v'.Id;
```

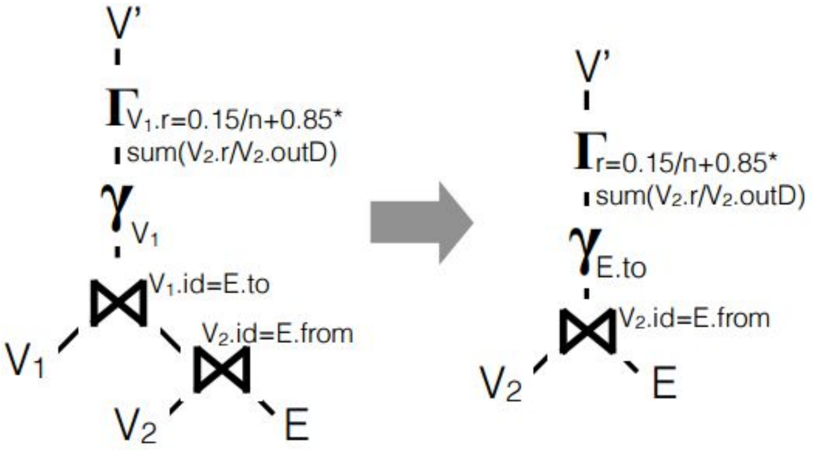
Query Optimizations

2) Incremental Evaluation

```
CREATE TABLE v_update_prime AS
  SELECT v1.id, MIN(v2.d+1) AS d
    FROM v_update AS v2, edge AS e, vertex AS v1
    WHERE v2.id=e.from_node AND v1.id=e.to_node
    GROUP BY e.to_node, v1.d
    HAVING MIN(v2.d+1) < v1.d;

DROP TABLE v_update;
ALTER TABLE v_update_prime RENAME TO v_update ;

CREATE TABLE vertex_prime AS
  SELECT v.id, ISNULL(v_update.d, v.d) AS value
    FROM vertex AS v LEFT JOIN v_update
    ON v.id = v_update.id;

DROP TABLE vertex; ALTER TABLE vertex_prime RENAME TO vertex;
```

Query Optimizations

2) Join Elimination



Join Elimination in PageRank

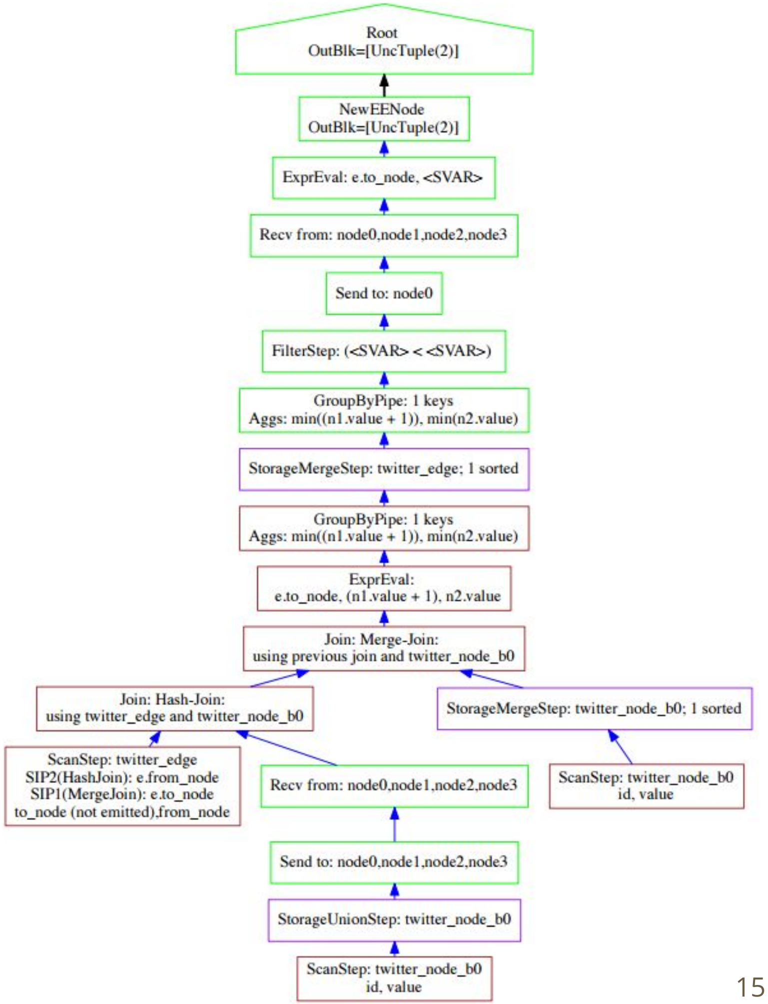
Query Execution

- Physical Design
  - Encoding and compression, sort orders, multiple table projections
- Join Optimization
  - Join directly over compressed data, choose from hash join and merge join
- Query Pipelining
  - Avoids materializing intermediate output and repeated access to disk
- Intra-query Parallelism
  - Process subgraphs in parallel across cpu cores using GroupBy

Query Execution Plan of SSSP

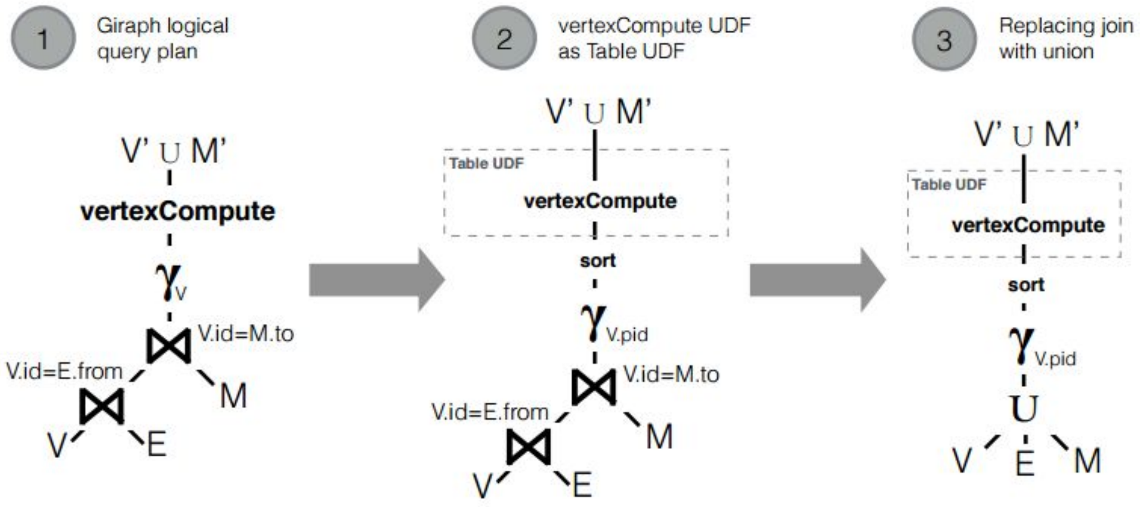
Different from Giraph execution pipeline:

1. Filter unnecessary tuples as early as possible.
2. Fully pipelines the execution flow.
3. Picks the best join execution strategy.



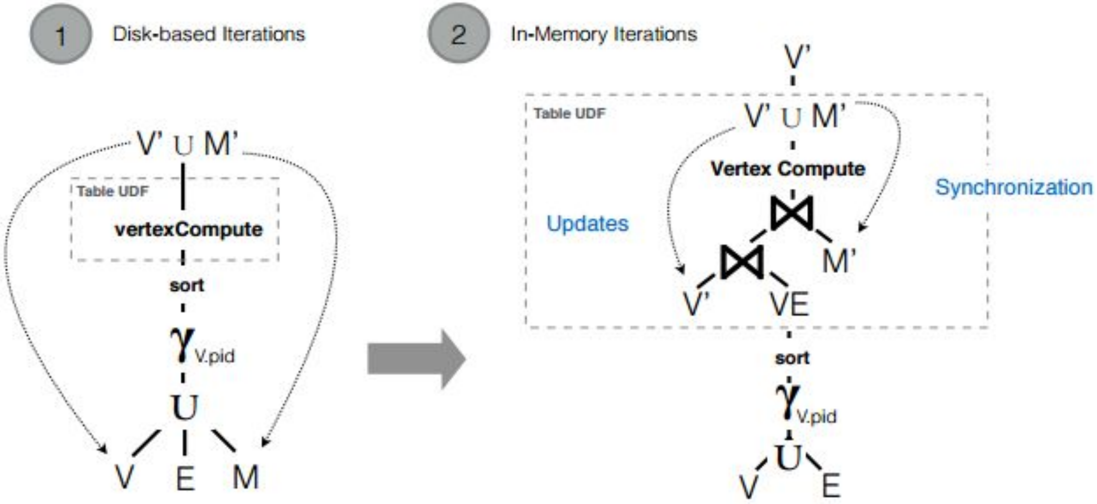
Extending Vertica

- Running unmodified vertex programs
  - As table UDFs without translating to relational operators



Extending Vertica

- Avoiding Intermediate Disk I/O
  - Load and store graph in shared memory, higher memory footprint



Experiments

Setup:

- Cluster of 4 machines
- 48 GB memory
- 1.4 TB Disk

Dataset:

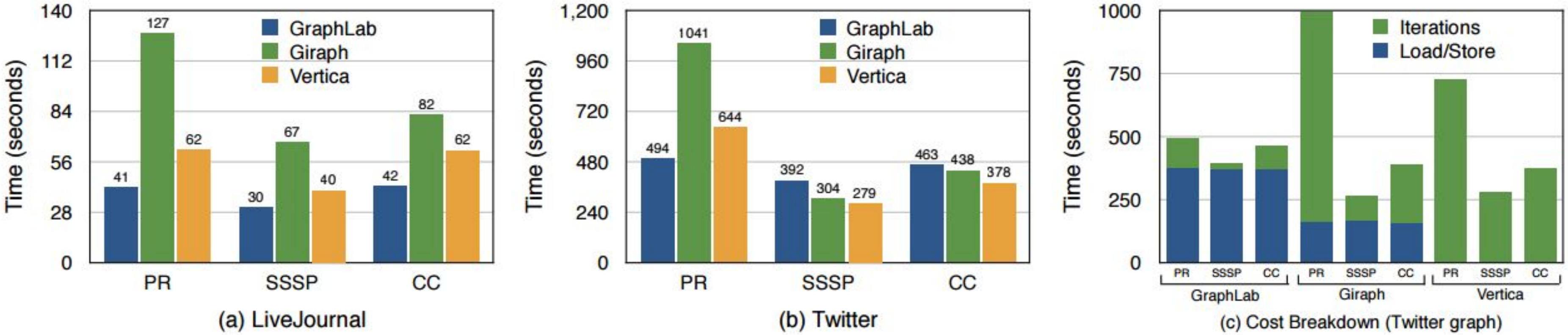
Type	Name	Nodes	Edges
Directed	Twitter-small	81,306	1,768,149
	LiveJournal	4,847,571	68,993,773
	Twitter	41,652,230	1,468,365,182
Undirected	YouTube	1,134,890	2,987,624
	LiveJournal-undir	3,997,962	34,681,189

Experiments

Data Preparation:

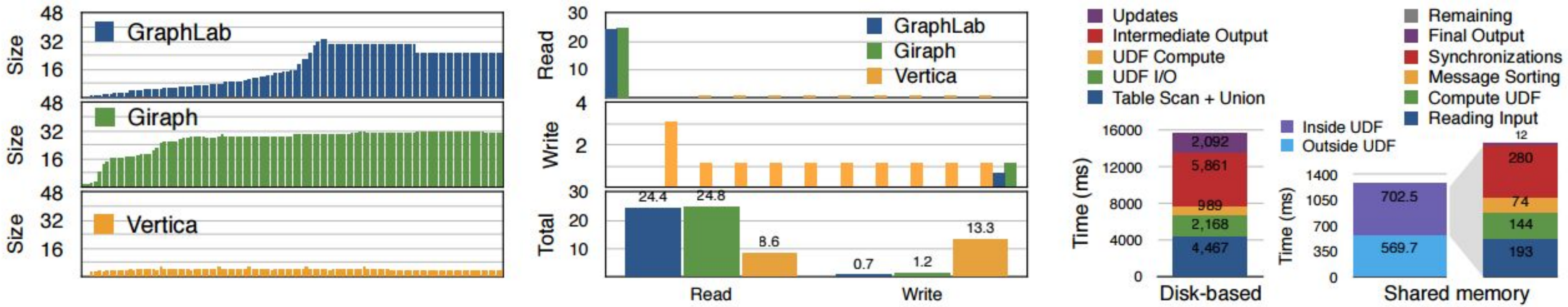
Metric	Dataset	Vertica	GraphLab	Giraph
Upload Time (sec)	LiveJournal	45.927	15.621	12.049
	Twitter	916.421	472.358	267.799
Disk Usage (GB)	LiveJournal	0.423	3.030	3.030
	Twitter	9.964	73.140	73.140

Runtime:



Experiments

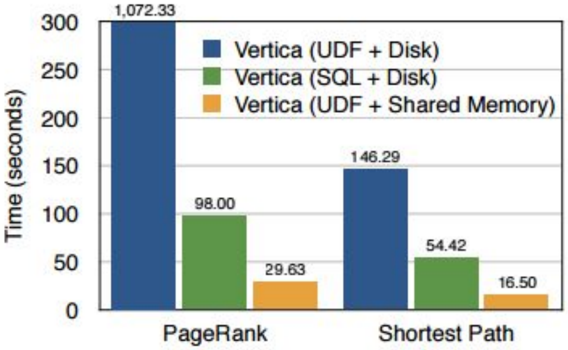
Memory Usage (PageRank):



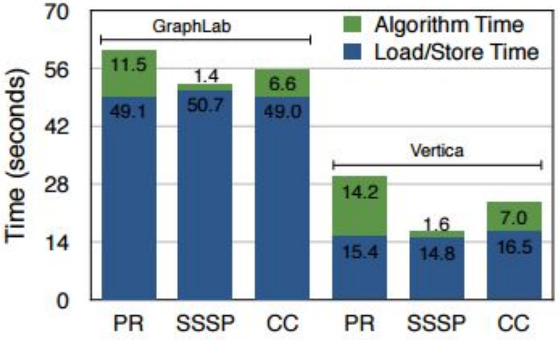
(a) Memory Footprint (GB) (b) I/O Footprint (GB) (c) UDF Cost Breakdown

Experiments

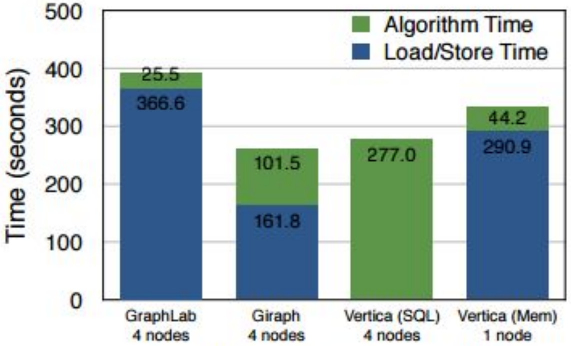
In memory Graph Analysis:



(a) Comparing different implementations on Vertica (LiveJournal graph)



(b) Comparing in-memory computation in GraphLab and Vertica (LiveJournal graph)



(c) Scaling and comparing Vertica in-memory on Twitter graph (SSSP)

Experiments

Mixed Graph and Relational Analysis :

Query	Dataset	Vertica	Giraph	SpeedUp
Sub-graph Projection & Selection	PR	55.6	954.6	17.2
	SSSP	101.3	405.5	4.0
Graph Analysis Aggregation	PR	643.9	1089.7	1.7
	SSSP	279.8	349.9	1.3
Graph Joins	PR+SSSP	927.0	1435.9	1.5

More Complicated Graph Processing:

Query	Dataset	Vertica	Giraph
Strong Overlap	Youtube	259.56	230.01
	LiveJournal-undir	381.05	out of memory
Weak Ties	Youtube	746.14	out of memory
	LiveJournal-undir	1,475.99	out of memory

Conclusion

- Vertica can be tuned to offer good end-to-end performance on graph queries (because it is optimized for scans, joins and aggregates).

- Users can trade memory with reduced I/O cost in iterative graph analysis.

- Relational databases can combine graph processing with relational analysis as pre-processing or post-processing steps.

- Features of relational databases can be combined with graph processing systems and it might be a good idea to stitch these systems together.

Thank you for your attention.