GraphLab: Graph-based distributed computation framework

Yihan Gao

University of Illinois at Urbana-Champaign

October 23, 2013
Motivation

- Large Graph is infeasible to analyze on single machine:
  - Twitter
  - Wikipedia

- Solution: Distributed Computing
Distributed Computation Framework:
Abstraction which hides lower level details of parallel programming, make it easier to write distributed algorithms
Distributed Computation Frameworks

- Distributed Computation Framework: Abstraction which hides lower level details of parallel programming, make it easier to write distributed algorithms

- Some Popular Distributed Computation Frameworks:
  - Map-Reduce (Google, OSDI’04)
  - Spark (UC Berkeley)
  - Pregel (Google, SIGMOD’10)
  - GraphLab (CMU)
Map-Reduce: Key-Value pair based distributed computation framework (Dean & Ghemawat, OSDI’04)

Assumes independence among tasks.
GraphLab: Graph-based distributed computation framework (UAI 2010, VLDB 2012)

Exploits graph locality to achieve parallel computing.
The key to distributed computing in graph is to think like a vertex (Google, 2010).

Idea: each vertex computes individually its value (in parallel).

Each vertex can see its local context, and updates its value accordingly.
Example: PageRank

Pseudo-code:

To compute PageRank of page P:
acc = 0;
for each In-page Q do
    acc += Q.pagerank / Q.num_out_links;
end
P.pagerank = 0.85 * acc + 0.15;
Data Model in GraphLab

- Directed Graph: $G = (V, E)$
- Each edge and vertex associated with a value (user-defined type)
- Vertex and edge values can be modified

- vertex or edge data in scope of red vertex
GraphLab supports two execution models.
GraphLab supports two execution models.

- Bulk-Synchronous: All vertices update in parallel (at the same time).
GraphLab supports two execution models.

- Bulk-Synchronous: All vertices update in parallel (at the same time).
- Asynchronous: If two vertices are not connected with distance $\leq 2$, can update them in parallel.
Gather-Apply-Scatter

- In GraphLab, each vertex program is further decomposed into three phases: Gather, Apply, Scatter.
In GraphLab, each vertex program is further decomposed into three phases: Gather, Apply, Scatter.

In Gather phase, the `gather()` function is executed in every edge adjacent to the vertex.
In GraphLab, each vertex program is further decomposed into three phases: Gather, Apply, Scatter.

In Gather phase, the `gather()` function is executed in every edge adjacent to the vertex.

In Apply phase, the values returned by the gather’s are summed together and given to the `apply()` function.
In GraphLab, each vertex program is further decomposed into three phases: Gather, Apply, Scatter.

In Gather phase, the `gather()` function is executed in every edge adjacent to the vertex.

In Apply phase, the values returned by the gather’s are summed together and given to the `apply()` function.

In Scatter phase, `scatter()` function is once again executed in each of the vertex’s adjacent edges.
PageRank in GAS

Gather

total = 0;
for \( j \in \text{in}\_\text{neighbors}(i) \) do
  total = total + \( R[j] \times w_{ji} \)
end

Apply

\( R[i] = 0.1 + \text{total} \);

Scatter

If \( R[i] \) not converged then
for \( j \in \text{out}\_\text{neighbors}(i) \) do
  signal vertex-program on \( j \);
end
To write a vertex program using GAS, we need to define a class extending `graphlab::ivertex_program<graph_type,gather_type>`.
To write a vertex program using GAS, we need to define a class extending `graphlab::ivertex_program<graph_type,gather_type>`.

There are five functions that need to be implemented: `gather_edges`, `gather`, `apply`, `scatter_edges`, `scatter`. 
GAS Interface – Gather

**gather_edges(vertex)**

- Specify the type of edges to be gathered.
- Return `graphlab::IN_EDGES`, `graphlab::OUT_EDGES`, `graphlab::NO_EDGES`, or `graphlab::ALL_EDGES`.

**gather(vertex, edge)**

- Return the value to be gathered.
- All the return values gathered will be *accumulated* and used in apply phase.
apply(vertex, total)

- The **sum of returned value** in gather phase are passed to apply function as parameter.
- Update the vertex data accordingly.
GAS Interface – Scatter

**scatter_edges(vertex)**

- Specify the type of edges to be scattered.
- Similar to gather_edges.

**scatter(vertex, edge)**

- Modify edge data or signal the other vertex.
- Similar to gather function except no return value is needed.
Latest feature in GraphLab.
Warp System

- Latest feature in GraphLab.
- `warp::parfor_all_vertices()` : simple parallel for loop over vertices.
Warp System

- Latest feature in GraphLab.
- `warp::parfor_all_vertices()`: simple parallel for loop over vertices.
- `warp::map_reduce_neighborhood()`: map-reduce aggregation over the neighborhood of vertex.
Warp System

- Latest feature in GraphLab.
- `warp::parfor_all_vertices()` : simple parallel for loop over vertices.
- `warp::map_reduce_neighborhood()` : map-reduce aggregation over the neighborhood of vertex.
- `warp::transform_neighborhood()` : parallel transformation of the neighborhood of vertex.
Warp System

- Latest feature in GraphLab.
- `warp::parfor_all_vertices()` : simple parallel for loop over vertices.
- `warp::map_reduce_neighborhood()` : map-reduce aggregation over the neighborhood of vertex.
- `warp::transform_neighborhood()` : parallel transformation of the neighborhood of vertex.
- `warp::broadcast_neighborhood()` : similar to `warp::transform_neighborhood()` but can also signal other vertices.
PageRank in Warp System

Main Function

```cpp
for (int i = 0; i < NUM_ITER; ++i)
    warp::parfor_all_vertices(graph,pagerank);
```

Update Function

```cpp
float pagerank_map(graph_type::edge_type edge,
                    graph_type::vertex_type other) {
    return other.data() / other.num_out_edges();
}

void pagerank(graph_type::vertex_type vertex) {
    vertex.data() = 0.15 +
    0.85 * warp::map_reduce_neighborhood(vertex,
                                           IN_EDGES, pagerank_map);
}
PageRank in Warp System

Main Function

```cpp
for (int i = 0; i < NUM_ITER; ++i)
    warp::parfor_all_vertices(graph, pagerank);
```

Update Function

```cpp
float pagerank_map(graph_type::edge_type edge,
                    graph_type::vertex_type other) {
    return other.data() / other.num_out_edges();
}

void pagerank(graph_type::vertex_type vertex) {
    vertex.data() = 0.15 +
    0.85 * warp::map_reduce_neighborhood(vertex,
    IN_EDGES, pagerank_map);
}```
Asynchronous version of PageRank can be implemented similarly using `warp::broadcast_neighborhood()`
Asynchronous version of PageRank can be implemented similarly using `warp::broadcast_neighborhood()`.

Gather type could be user-defined type, with “+=” operation implemented.
Asynchronous version of PageRank can be implemented similarly using `warp::broadcast_neighborhood()`.

Gather type could be user-defined type, with “+=” operation implemented.

GraphLab is implemented in C++, but there are projects porting GraphLab to Python.
GraphLab Official Website: www.graphlab.org

Introduction to Large-Scale Graph Computation (Aapo Kyrola, 2013)

PowerGraph: Distributed graph-parallel computation on natural graphs (Joseph Gonzalez, OSDI’12)

MapReduce & GraphLab: Programming Models for Large-Scale Parallel/Distributed Computing (Iftekhar Naim, 2013)