COMPSCI 590S Systems for Data Science **Fall 2017**

Lecture 12: Graph Storage

Lecturer: Emery Berger *Scribe(s): Ariel Reches, Saranya Krishnakumar*

12.1 Representing Graphs

How can we represent graphs, how do they fit into any of the previous big data storage frameworks we learned?. To work on graphs using previous data storage systems, graphs had to be represented in a matrix or list form and then worked upon.

1. Database systems we know :
   (a) NoSQL, Map Redpuce, Key Values
   (b) SQL, Reg Databases, tuples
   (c) OOdatabases, "objects", XML for example
2. Representing Graphs :
   (a) Ways to represent :
      i. Adjacency Matrix - symmetric along diagonal for undirected graph, hence only half of the matrix needs to be stored, $O(V^2)$
      ii. Adjacency List - vertex edge notation, $O(V + E)$ ex. 1(2,3) 4(5,6)
      iii. (Edge, Edge) - every edge is represented as set of vertices, $O(2E)$ ex. (1,2) (1,3) (4,5) (4,6)
   (b) Degrees of Freedom :
      i. Directed/Undirected - when connections are symmetric undirected graphs are used, when connections are asymmetric directed graphs are used for representing(Twitter relationship vs Facebook relationship)
      ii. Static vs Dynamic (Points on a map vs friends in a social network)

12.2 Sparse Graphs

A sparse graph is one in which many pairs of vertices do not share any edges. An adjacency matrix is an inefficient way to hold sparse graphs even considering the benefit that all graph operations can be turned to matrix operations. Most of the graphs in general are sparse, ex. in social networks like Facebook, there are lesser than N^2 edges, not everyone is friends with everyone else. Complete graphs are unlikely in reality. Hence we need optimal way of representing sparse matrices.

1. Matrix representation bad for sparse data
2. Run Length encoding ex. (1, 1, 4, 1) = starting at 1,1 there are 4 1s in a row
   (a) Good for sparse matrices generally but usually doesn't work for graphs
3. Sparse graph usually uses adjacency list $O(V + E)$ or (vertex, vertex) $O(2E)$

12.3 Map Reduce with Edge List

Map Reduce with Edge Lists is bad for a few reasons

1. Too much IO (many Map steps involving disk write/ disk read)

2. Close edges likely to be widely distributed across cluster(poor locality)

3. Map Reduce completely oblivious to the graph structure

12.4 Graph Computation

Previously all graph computations were run in one machine. But this was not efficient for processing large graphs. Hence distributing computation across many machines was introduced in Pregel. Pregel is a vertex centric model. All edges with a vertex are stored locally along with the vertex, so that no message passing is required between vertex and edge for computation, also gives fault tolerance. Pregel does not work when there are too many edges associated with a vertex (ex. Jutin Bieber and Ellen Degeneres problem).

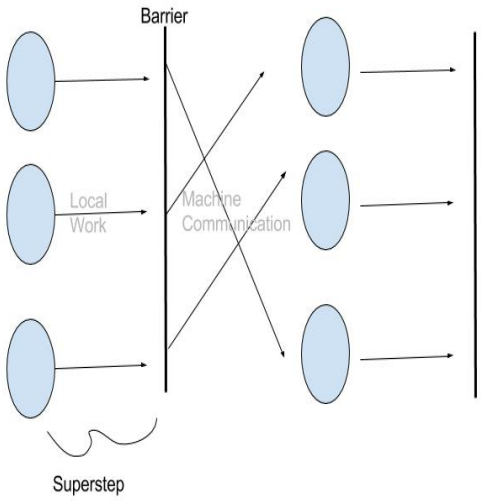
12.5 BSP + MPI

BSP(Bulk Synchronous Parallel) is an alternative general synchronized programming model that is used implemented by Pregel, the Map Reduce alternative for Graphs. The communications between machines implements MPI (Message Passing Interface).

1. We want locality

2. Parallelism

3. There's the Ellen Problem (too many edges on a single node = load imbalance or not able to fit in memory)



4. Bulk Synchronous Programming
   (a) Originates from PRAM a theoretic programming model assuming small unit costs of memory transaction (doesn't work in practice)
   (b) Operates in supersteps where all machines do local work. All machines must wait for the other ones to finish their superstep before crossing the "barrier" to communicate with other machines
   (c) Barriers aren't ideal (machines sit idly) but there is synchronism without deadlocks and races
   (d) For graphs, vertices and their edges can be split across machines
5. Message Passing Interface
   (a) Broadcast: send messages to all
   (b) Group communication : send message to some
   (c) Problems that could arise are
      i. Can easily end up in deadlocks
      ii. There are chances of communication race happening. This leads to non determinism, ex:

```
if mpi_self == 0
    mpi_recv:(ANY)
    print result
else
    mpi_send(self, 0)
```

   Each time the program is run, not same value may be printed.
6. Though synchronization with barriers can be slow, BSP model is preferred. We want determinism and no deadlocks. BSP addresses all these problems. Communication cycles are not possible in BSP since all sends happen followed by all receives.
7. When there are many edges associated with one vertex and few edges with all other vertices, then superstep time can be prolonged because of this one vertex. Random selection is better for load balancing.