Graph Processing

graph - hierarchical data structure modelling dependency relationship

Representation

graph (G) comprises nodes (V) and edges (E)

Adjacency matrix: n x n matrix M(n=|V|), non-zero Mij means edge from Vi to Vj

Adjacency list: List[(V, List[V])], tuple represents node and its connections

Edge list: List[(V, V)], node pairs representing edge

Structures

Graph components - subgraphs, any two vertices connected by path

Strongly connected comp. - largest sub-graph with path between any two nodes

Triangles/Polygons - three vertices connected to one another

Spanning tree - sub-graph of all nodes and minimum number of edges

Algorithms

Traversal - starting from node, find all connected nodes

  Depth-first - recursively follow edges until visiting all reachable nodes

  Breadth-first - follow graph edges per level

Node importance - calculate importance of node relative to other nodes

  Centrality measures or PageRank

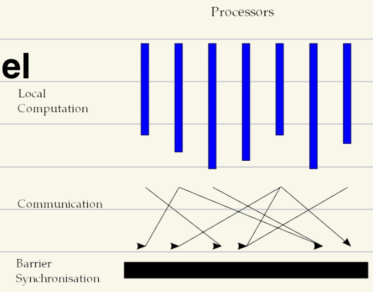
Shortest paths

  Dijkstra's algorithm or 'travelling salesman'

Graph DB - specialised RDBMs storing recursive data structures, supporting CRUD operations on them and maintaining transactional consistency

BSP (bulk synchronous parallel) model

- multiple processors with fast local memory
- pair-wise communication between processors
- barrier implementation synchronising super steps

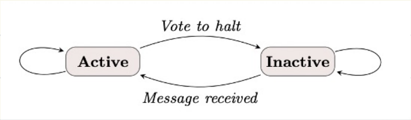


BSP superstep

- local execution: use own memory to compute on local data partitions
- data exchange/remote communication: data between processes
- barrier synchronisation: processes wait until all finished communicating

Termination

- Superstep 0: all vertices are active
- All active vertices participate in computation at superstep
- Vertex deactivates by voting to halt
- No execution in subsequent supersteps
- Vertex reactivated by receiving a message



Pregel (Google)

- distributed graph processing framework

- computations = sequence of supersteps
- superstep - framework invokes user-defined function for each vertex
- function specifies behaviour at a single vertex (V) and a single superstep (S)
  - read message sent to V in superstep (S-1)
  - send messages to other vertices, to be read at superstep (S+1)
  -modify state of V and outgoing edges

Vertex centric approach

- Reminiscent of MapReduce
  - user focus on local action
  - each vertex processed independently
- well suited for distributed implementation
  - communication from S to S+1
  - no defined execution order within superstep
  - free of deadlocks and data races

Roles

graphs - adjacency lists, partitioned and distributed using network file system

- leader - mapping between data partitions and cluster node; barrier
- worker - maintains for each vertex
  - adjacency list
  - current calculation value
  - queue of incoming messages
  - state

Superstep

- Workers combine incoming messages for all vertices
  - combinator function updates vertex state
- Termination condition met, vertex vote to exclude oneself
- Vertex updates global aggregator (optional)
- Message passing:
  - receiving vertex local: update message queue
  - else: wrap messages per receiving node, send in bulk

Fault tolerance

- reminiscent model of Spark
- periodically leader instruct worker to save in-memory state to storage
- leader - keep-alive messages to workers; failure detection
- failure - leader reassigns graph partitions to live workers