Pregel: A System for Large-Scale Graph Processing

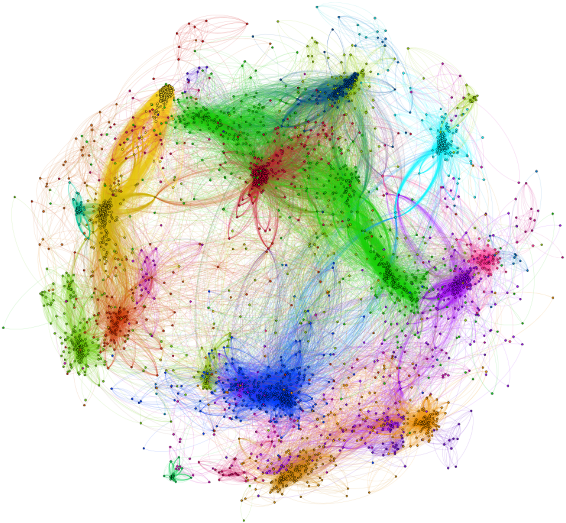
Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Introduction

- Graphs provide a flexible abstraction for describing relationships between discrete objects.
- Many problems can be modeled by graphs and solved with appropriate graph algorithms.

Large Graph



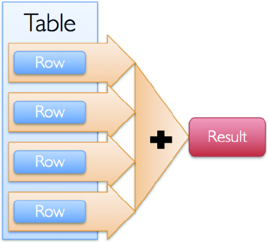
Large-Scale Graph Processing

- Large graphs need large-scale processing.

- A large graph either cannot fit into memory of single computer or it fits with huge cost.

Question

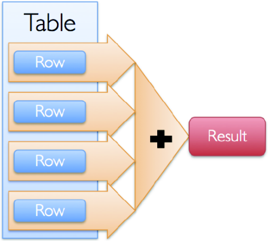
Can we use platforms like MapReduce or Spark, which are based on data-parallel model, for large-scale graph proceeding?



Data-Parallel Model for Large-Scale Graph Processing

- The platforms that have worked well for developing parallel applications are not necessarily effective for large-scale graph problems.

- Why?



Graph Algorithms Characteristics (1/2)

- Unstructured problems
  - Difficult to extract parallelism based on partitioning of the data: the irregular structure of graphs.
  - Limited scalability: unbalanced computational loads resulting from poorly partitioned data.

Graph Algorithms Characteristics (1/2)

- Unstructured problems
  - Difficult to extract parallelism based on partitioning of the data: the irregular structure of graphs.
  - Limited scalability: unbalanced computational loads resulting from poorly partitioned data.

- Data-driven computations
  - Difficult to express parallelism based on partitioning of computation: the structure of computations in the algorithm is not known a priori.
  - The computations are dictated by nodes and links of the graph.

Graph Algorithms Characteristics (2/2)

- Poor data locality
  - The computations and data access patterns do not have much locality: the irregular structure of graphs.

Graph Algorithms Characteristics (2/2)

- Poor data locality
  - The computations and data access patterns do not have much locality: the irregular structure of graphs.

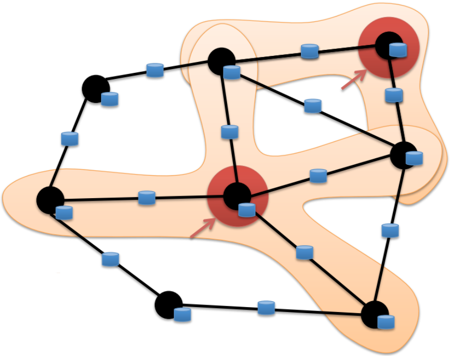
- High data access to computation ratio
  - Graph algorithms are often based on exploring the structure of a graph to perform computations on the graph data.
  - Runtime can be dominated by waiting memory fetches: low locality.

Proposed Solution

Graph-Parallel Processing

Proposed Solution

Graph-Parallel Processing

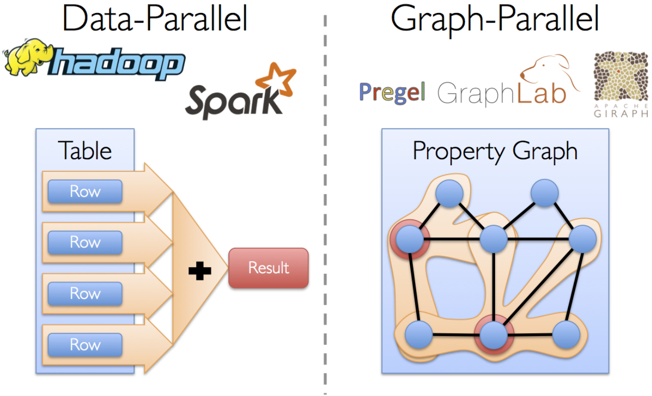


- Computation typically depends on the neighbors.

Graph-Parallel Processing

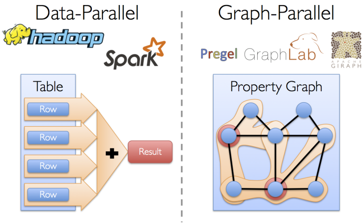
- Restricts the types of computation.
- New techniques to partition and distribute graphs.
- Exploit graph structure.
- Executes graph algorithms orders-of-magnitude faster than more general data-parallel systems.

Data-Parallel vs. Graph-Parallel Computation



Data-Parallel vs. Graph-Parallel Computation

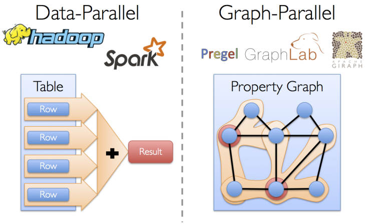
- Data-parallel computation
  - Record-centric view of data.
  - Parallelism: processing independent data on separate resources.



Data-Parallel vs. Graph-Parallel Computation

- Data-parallel computation
  - Record-centric view of data.
  - Parallelism: processing independent data on separate resources.

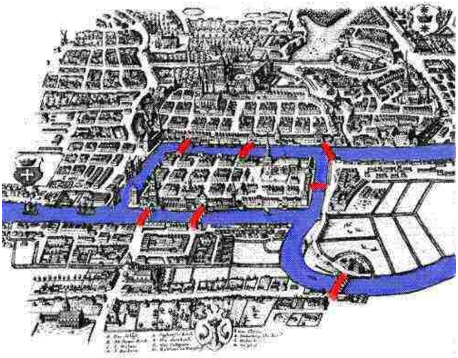
- Graph-parallel computation
  - Vertex-centric view of graphs.
  - Parallelism: partitioning graph (dependent) data across processing resources, and resolving dependencies (along edges) through iterative computation and communication.



Pregel
oogle

Seven Bridges of Königsberg

- Finding a walk through the city that would cross each bridge once and only once.
- Euler proved that the problem has no solution.



Map of Königsberg in Euler's time, highlighting the river Pregel and the bridges.

Pregel

- Large-scale graph-parallel processing platform developed at Google.
- Inspired by bulk synchronous parallel (BSP) model.

Bulk Synchronous Parallel (1/2)

- It is a parallel programming model.
- The model consists of:

Bulk Synchronous Parallel (1/2)

- It is a parallel programming model.
- The model consists of:
  - A set of processor-memory pairs.

Bulk Synchronous Parallel (1/2)

- It is a parallel programming model.

- The model consists of:
  - A set of processor-memory pairs.
  - A communications network that delivers messages in a point-to-point manner.

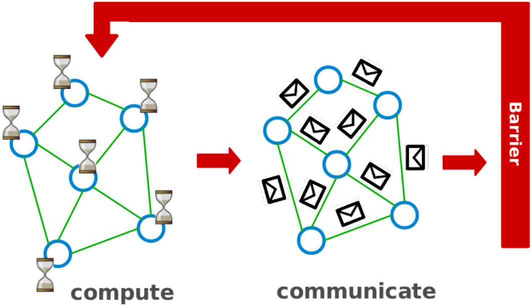
Bulk Synchronous Parallel (1/2)

- It is a parallel programming model.
- The model consists of:
  - A set of processor-memory pairs.
  - A communications network that delivers messages in a point-to-point manner.
  - A mechanism for the efficient barrier synchronization for all or a subset of the processes.

Bulk Synchronous Parallel (1/2)

- It is a parallel programming model.
- The model consists of:
  - A set of processor-memory pairs.
  - A communications network that delivers messages in a point-to-point manner.
  - A mechanism for the efficient barrier synchronization for all or a subset of the processes.
  - There are no special combining, replicating, or broadcasting facilities.

Bulk Synchronous Parallel (2/2)



All vertices update in parallel (at the same time).

Vertex-Centric Programs

- Think as a vertex.

Vertex-Centric Programs

- Think as a vertex.
- Each vertex computes individually its value: in parallel

Vertex-Centric Programs

- Think as a vertex.
- Each vertex computes individually its value: in parallel
- Each vertex can see its local context, and updates its value accordingly.

Data Model

- A directed graph that stores the program state, e.g., the current value.

Execution Model (1/3)

- Applications run in sequence of iterations: supersteps

Execution Model (1/3)

- Applications run in sequence of iterations: supersteps

- During a superstep, user-defined functions for each vertex is invoked (method `Compute()`): in parallel

Execution Model (1/3)

- Applications run in sequence of iterations: supersteps

- During a superstep, user-defined functions for each vertex is invoked (method `Compute()`): in parallel

- A vertex in superstep S can:
  - reads messages sent to it in superstep S-1.
  - sends messages to other vertices: receiving at superstep S+1.
  - modifies its state.

Execution Model (1/3)

- Applications run in sequence of iterations: supersteps
- During a superstep, user-defined functions for each vertex is invoked (method `Compute()`): in parallel
- A vertex in superstep S can:
  - reads messages sent to it in superstep S-1.
  - sends messages to other vertices: receiving at superstep S+1.
  - modifies its state.
- Vertices communicate directly with one another by sending messages.

Execution Model (2/3)

- Superstep 0: all vertices are in the active state.

Execution Model (2/3)

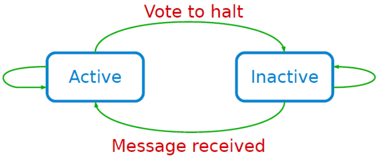
- Superstep 0: all vertices are in the active state.
- A vertex deactivates itself by voting to halt: no further work to do.

Execution Model (2/3)

- Superstep 0: all vertices are in the active state.
- A vertex deactivates itself by voting to halt: no further work to do.
- A halted vertex can be active if it receives a message.

Execution Model (2/3)

- Superstep 0: all vertices are in the active state.
- A vertex deactivates itself by voting to halt: no further work to do.
- A halted vertex can be active if it receives a message.
- The whole algorithm terminates when:
  - All vertices are simultaneously inactive.
  - There are no messages in transit.



Execution Model (3/3)

- Aggregation: a mechanism for global communication, monitoring, and data.

Execution Model (3/3)

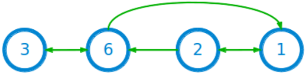
- Aggregation: a mechanism for global communication, monitoring, and data.
- Runs after each superstep.
- Each vertex can provide a value to an aggregator in superstep S.
- The system combines those values and the resulting value is made available to all vertices in superstep $S + 1$.

Execution Model (3/3)

- Aggregation: a mechanism for global communication, monitoring, and data.
- Runs after each superstep.
- Each vertex can provide a value to an aggregator in superstep S.
- The system combines those values and the resulting value is made available to all vertices in superstep $S + 1$.
- A number of predefined aggregators, e.g., min, max, sum.
- Aggregation operators should be commutative and associative.

Example: Max Value (1/4)

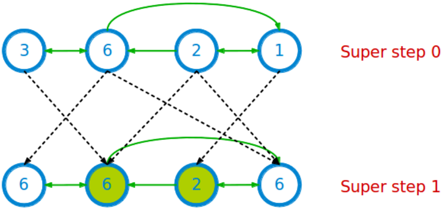
```
i_val := val

for each message m
  if m > val then val := m

if i_val == val then
  vote_to_halt
else
  for each neighbor v
    send_message(v, val)
```



Super step 0

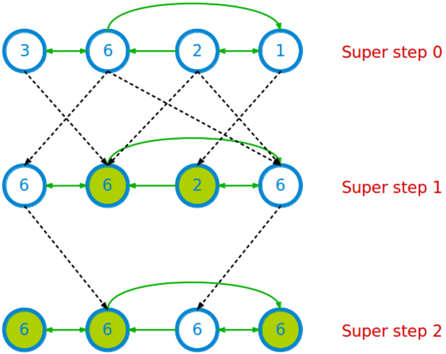
Example: Max Value (2/4)

```
i_val := val

for each message m
  if m > val then val := m

if i_val == val then
  vote_to_halt
else
  for each neighbor v
    send_message(v, val)
```



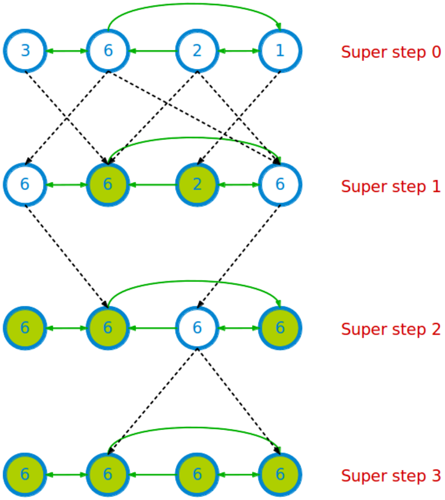
Example: Max Value (3/4)

```
i_val := val

for each message m
  if m > val then val := m

if i_val == val then
  vote_to_halt
else
  for each neighbor v
    send_message(v, val)
```



Example: Max Value (4/4)

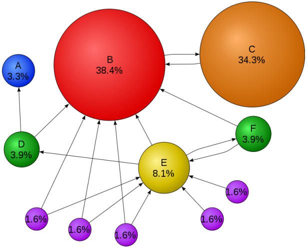
```
i_val := val

for each message m
  if m > val then val := m

if i_val == val then
  vote_to_halt
else
  for each neighbor v
    send_message(v, val)
```



Example: PageRank

- Update ranks in parallel.
- Iterate until convergence.

$$R[i] = 0.15 + \sum_{j \in Nbrs(i)} w_{ji} R[j]$$



Example: PageRank

```
Pregel_PageRank(i, messages):
  // receive all the messages
  total = 0
  foreach(msg in messages):
    total = total + msg

  // update the rank of this vertex
  R[i] = 0.15 + total

  // send new messages to neighbors
  foreach(j in out_neighbors[i]):
    sendmsg(R[i] * wij) to vertex j
```

$$R[i] = 0.15 + \sum_{j \in Nbrs(i)} w_{ji} R[j]$$

Partitioning the Graph

- The pregel library divides a graph into a number of partitions.
- Each consisting of a set of vertices and all of those vertices' outgoing edges.
- Vertices are assigned to partitions based on their vertex-ID (e.g., hash(ID)).

Implementation (1/4)

- Master-worker model.
- User programs are copied on machines.
- One copy becomes the master.

Implementation (2/4)

- The master is responsible for
  - Coordinating workers activity.
  - Determining the number of partitions.

- Each worker is responsible for
  - Maintaining the state of its partitions.
  - Executing the user's `Compute()` method on its vertices.
  - Managing messages to and from other workers.

Implementation (3/4)

- The master assigns one or more partitions to each worker.

Implementation (3/4)

- The master assigns one or more partitions to each worker.

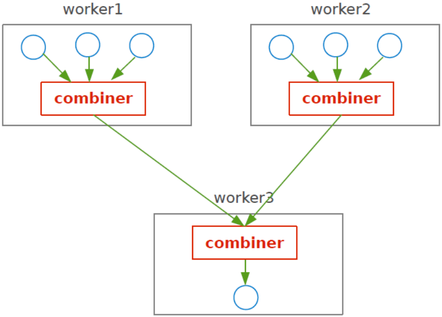
- The master assigns a portion of user input to each worker.
  - Set of records containing an arbitrary number of vertices and edges.
  - If a worker loads a vertex that belongs to that worker's partitions, the appropriate data structures are immediately updated.
  - Otherwise the worker enqueues a message to the remote peer that owns the vertex.

Implementation (4/4)

- After the input has finished loading, all vertices are marked as active.
- The master instructs each worker to perform a superstep.
- After the computation halts, the master may instruct each worker to save its portion of the graph.

Combiner

- Sending a message between workers incurs some overhead: use combiner.
- This can be reduced in some cases: sometimes vertices only care about a summary value for the messages it is sent (e.g., min, max, sum, avg).



Fault Tolerance (1/2)

- Fault tolerance is achieved through checkpointing.
- At start of each superstep, master tells workers to save their state:
  - Vertex values, edge values, incoming messages
  - Saved to persistent storage
- Master saves aggregator values (if any).
- This is not necessarily done at every superstep: costly

Fault Tolerance (2/2)

- When master detects one or more worker failures:
  - All workers revert to last checkpoint.
  - Continue from there.
  - That is a lot of repeated work.
  - At least it is better than redoing the whole job.

Pregel Limitations

- Inefficient if different regions of the graph converge at different speed.
- Can suffer if one task is more expensive than the others.
- Runtime of each phase is determined by the slowest machine.

Pregel Summary

- Bulk Synchronous Parallel model
- Vertex-centric
- Superstep: sequence of iterations
- Master-worker model
- Communication: message passing

Questions?