Large Scale Distributed Data Processing
Graphs, Vertex Processing, GraphX

Silviu Maniu

April 7th, 2022

Table of contents

Graphs and Algorithms

Graphs – data model: edges between vertices (with properties/data or not)

Graphs are everywhere where binary relations are important:

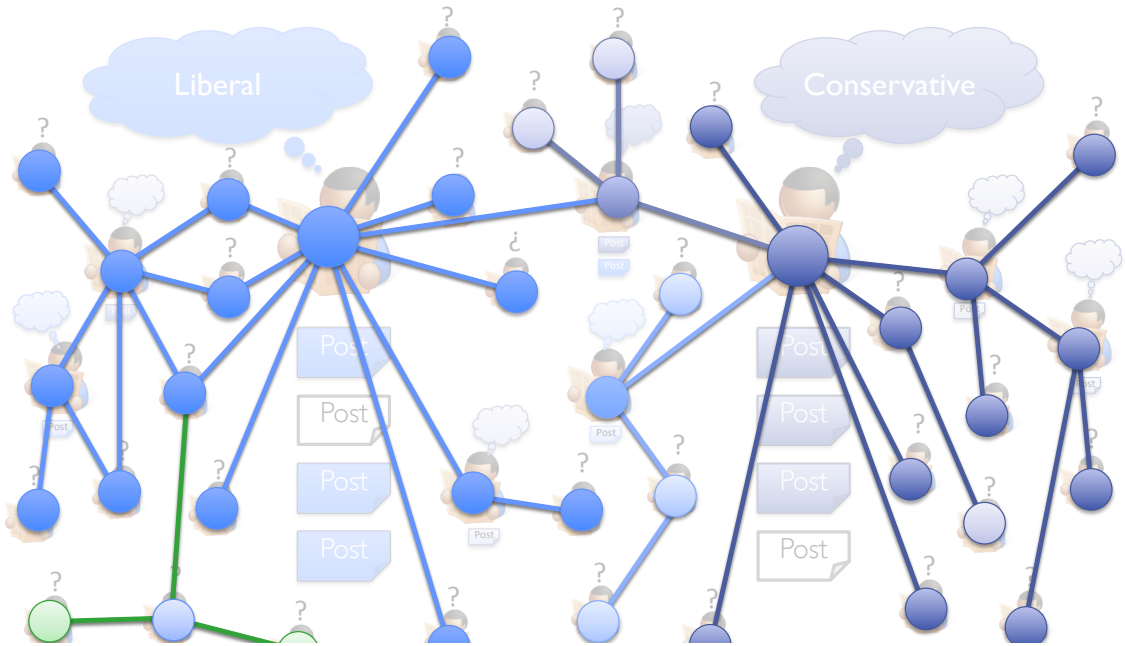
- social networks (friendships), Web graphs (links between machines), communication (network packets), transactions

Graphs and Algorithms

Graphs – data model: edges between vertices (with properties/data or not)

Graph algorithms are important:

- classic algorithms: shortest paths, community detection, counting triangles, PageRank
- data mining/machine learning on graphs: collaborative filtering, belief propagation, mean-field algorithms, graph neural networks



Distributing Graph Computation

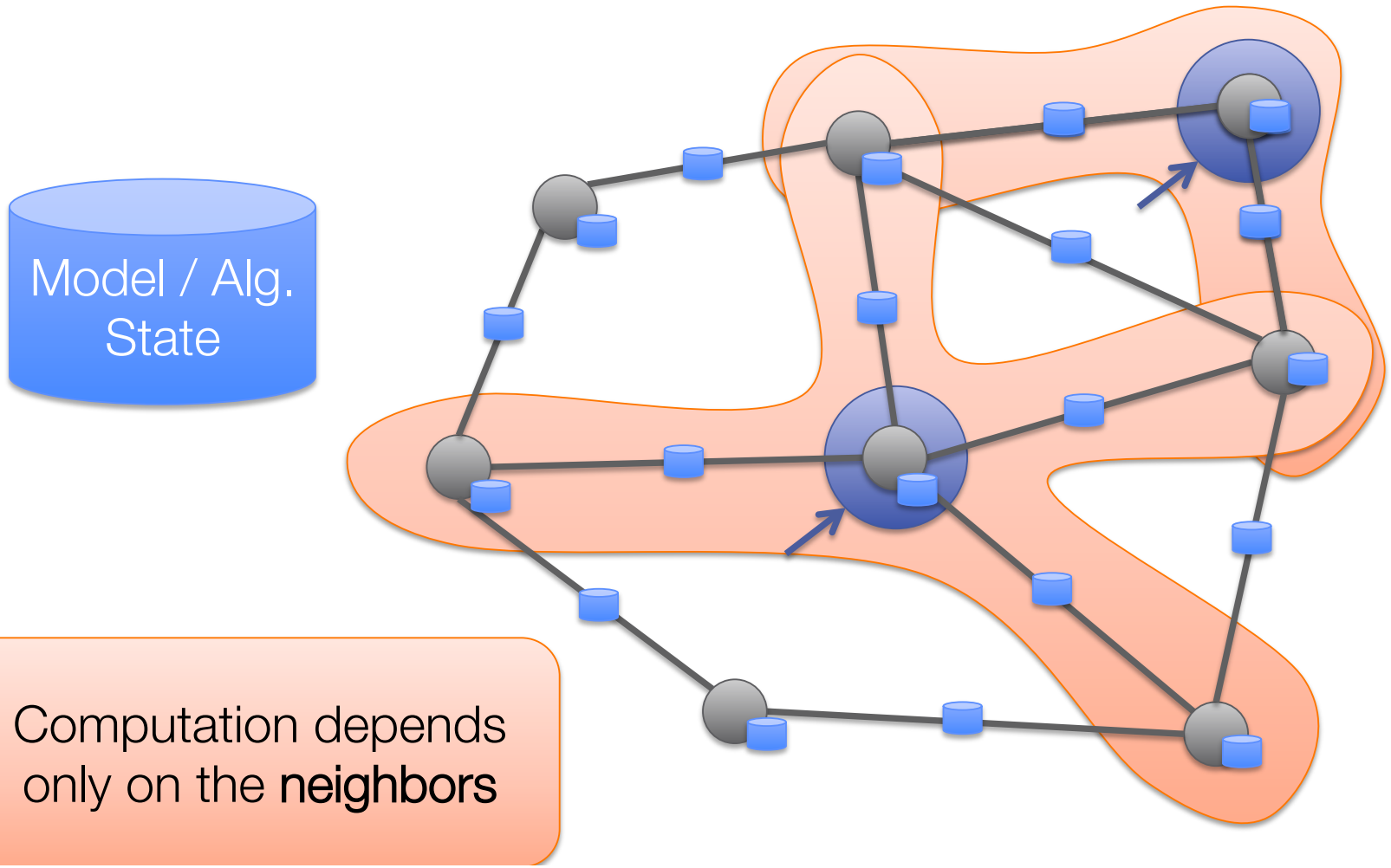
Challenges in efficiently processing graph algorithms:

- locality issues: since different vertices are needed
- little work per vertex: some vertices may not be used at all
- parallelism is not the same everywhere

MapReduce, Spark are **data parallel**: very efficient for tabular data, aggregating data – not the case for graphs

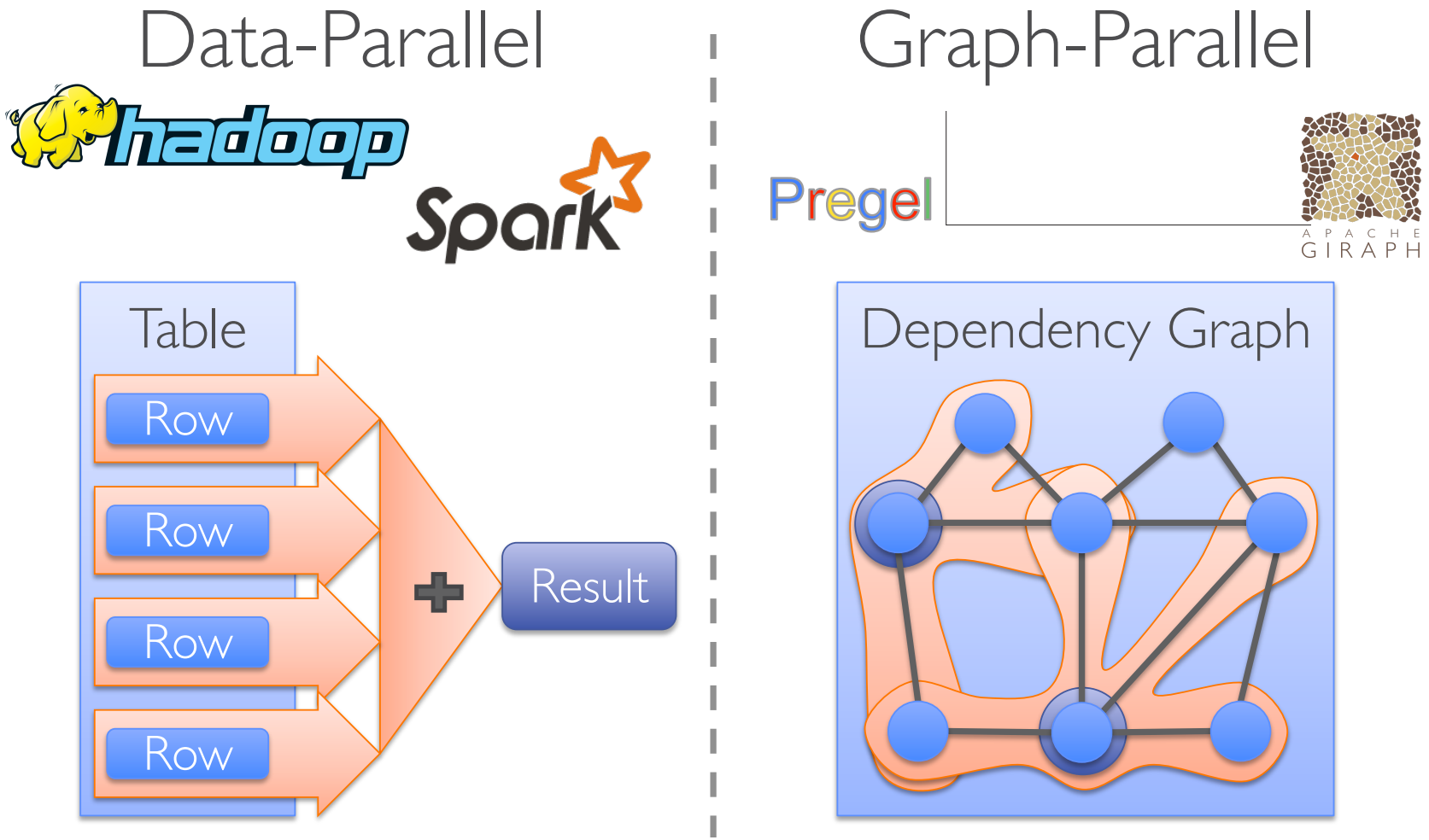
Distributing Graph Computation – Graph Parallel

Graph parallel – computations are made locally, via neighbours



from https://stanford.edu/~rezab/sparkclass/slides/ankur_graphx.pdf

Data Parallel vs. Graph Parallel



from `https://stanford.edu/~rezab/sparkclass/slides/ankur_graphx.pdf`

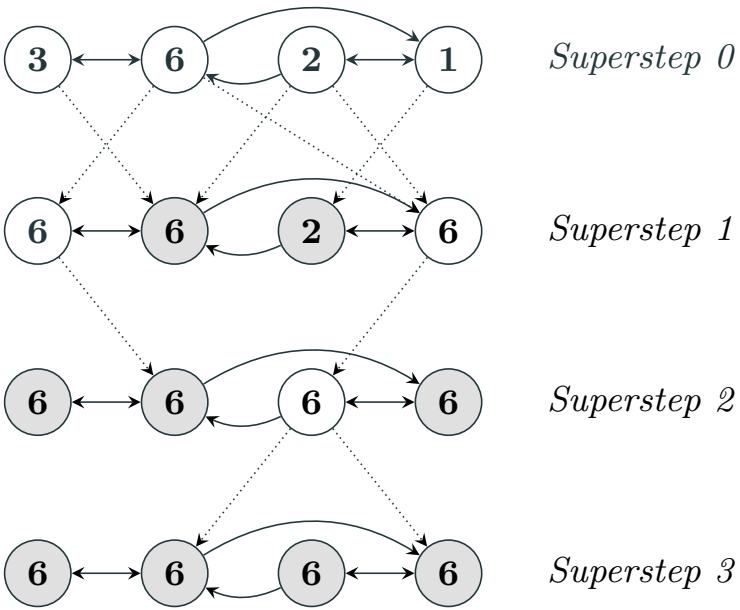
Graph Processing with Pregel

Google Pregel [Malewicz et al., 2010] – implements a variant of the Bulk Synchronous Processing model [Valiant, 1990]

Computation Model

1. input: each vertex in the graph receives initial data
2. supersteps with global synchronization points
   - each vertex receives the same *user-defined function* to execute in a superstep
   - between supersteps, messages are sent between vertices (either via links, or globally)
   - in a superstep, each vertex can modify its state, and set-up a message to be sent to other vertices
   - vertices can vote to halt the computation; if they receive new messages they must re-activate
3. computation stops when all vertices vote to halt and are not reactivated

Vertex Processing with Pregel – Max Value Example and API




```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

from [Malewicz et al., 2010]

Pregel Operations – PageRank Example

PageRank [Brin and Page, 1998] – well-known algorithm for computing most important nodes in a graph

- iterative (and recursive): the rank of the node depends on the ranks of the neighbours

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

Rank of user *i*

Weighted sum of neighbors' ranks

```
def PageRank(v: Id, msgs: List[Double]) {
  // Compute the message sum
  var msgSum = 0
  for (m <- msgs) { msgSum += m }
  // Update the PageRank
  PR(v) = 0.15 + 0.85 * msgSum
  // Broadcast messages with new PR
  for (j <- OutNbrs(v)) {
    msg = PR(v) / NumLinks(v)
    send_msg(to=j, msg)
  }
  // Check for termination
  if (converged(PR(v))) voteToHalt(v)
}
```

from `https://stanford.edu/~rezab/sparkclass/slides/ankur_graphx.pdf` and [Gonzalez et al., 2014]

Pregel – Architecture

Partition by vertex id – set of vertices and all outgoing edges

- assignment method to machines can vary greatly

Master architecture: machine assigned from program copies that are sent to the cluster

- decides partitioning and resulting worker machines, checks halting votes

Fault tolerance: using checkpointing mechanism

- when a worker fails: master re-assigns graph partitions, which is resumed from the latest checkpoint

Gather – Scatter – Apply (GraphLab)

GAS decomposition [Gonzalez et al., 2012] – 3 data-parallel stages:

1. gather: aggregate the messages from neighbours
2. apply: execute the UDF in-vertex
3. scatter: send the messages to neighbours

Better parallelism than general Pregel (vertex-cut partitioning), but does not allow communication between vertices that are not neighbours

PageRank

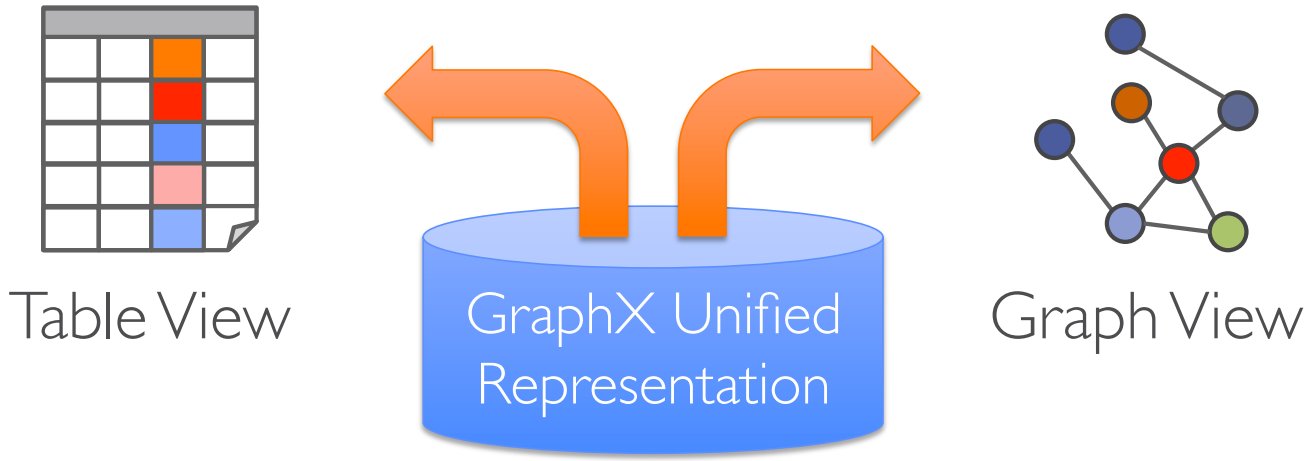
```
def Gather(a: Double, b: Double) = a + b
def Apply(v, msgSum) {
  PR(v) = 0.15 + 0.85 * msgSum
  if (converged(PR(v))) voteToHalt(v)
}
def Scatter(v, j) = PR(v) / NumLinks(v)
```

Table of contents

GraphX [Gonzalez et al., 2014]

- built on top of Spark 

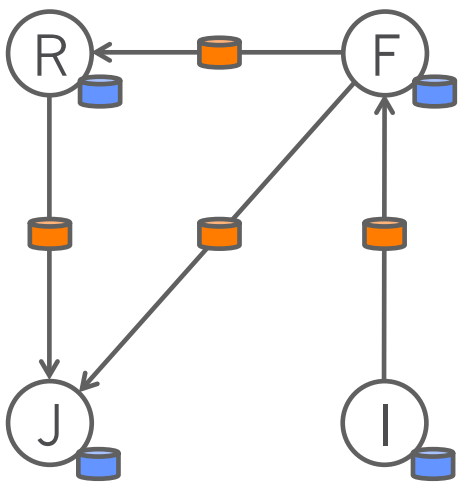
- unifies the table view and the graph view
- unifies data-parallel and graph-parallel
- uses the Spark RDD operators for the table view, new operators for graph view



Property Graph

Data Model: property graph (attributes on edges and vertices), seen as a graph

Property Graph



Vertex Table

Id	Attribute (V)
Rxin	(Stu., Berk.)
Jegonzal	(PstDoc, Berk.)
Franklin	(Prof., Berk)
Istoica	(Prof., Berk)

Edge Table

SrcId	DstId	Attribute (E)
rxin	jegonzal	Friend
franklin	rxin	Advisor
istoica	franklin	Coworker
franklin	jegonzal	PI

Originaal GraphX Graph Operators

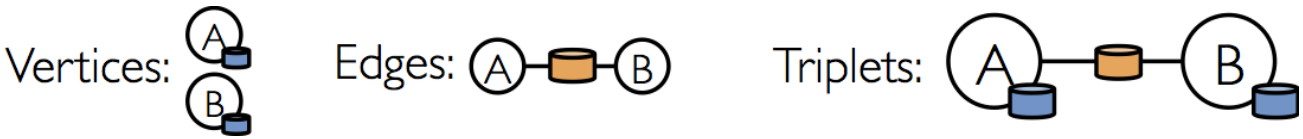
```
class Graph[V, E] {
  // Constructor
  def Graph(v: Collection[(Id, V)],
            e: Collection[(Id, Id, E)])
  // Collection views
  def vertices: Collection[(Id, V)]
  def edges: Collection[(Id, Id, E)]
  def triplets: Collection[Triplet]
  // Graph-parallel computation
  def mrTriplets(f: (Triplet) => M,
      sum: (M, M) => M): Collection[(Id, M)]
  // Convenience functions
  def mapV(f: (Id, V) => V): Graph[V, E]
  def mapE(f: (Id, Id, E) => E): Graph[V, E]
  def leftJoinV(v: Collection[(Id, V)],
      f: (Id, V, V) => V): Graph[V, E]
  def leftJoinE(e: Collection[(Id, Id, E)],
      f: (Id, Id, E, E) => E): Graph[V, E]
  def subgraph(vPred: (Id, V) => Boolean,
      ePred: (Triplet) => Boolean)
    : Graph[V, E]
  def reverse: Graph[V, E]
}
```

The latest version is available at `https://spark.apache.org/docs/latest/graphx-programming-guide.html`

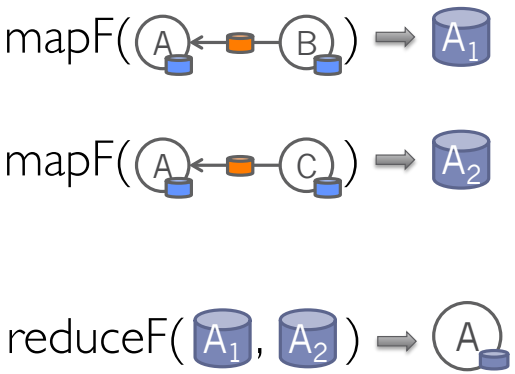
mrTriplets Operator (Legacy)

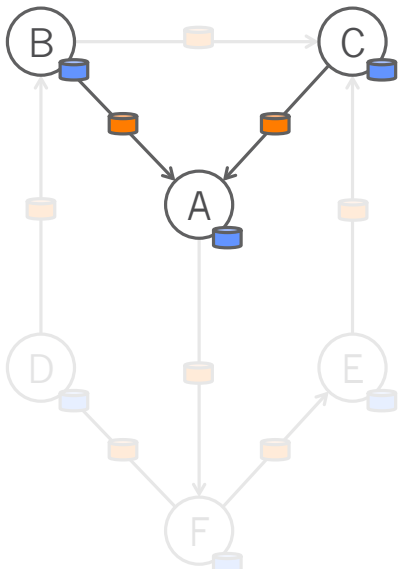
Previous versions of GraphX provided operators for MapReduce over triples of edges and vertices – resulting from the join of the vertex and edge table

```
SELECT src.Id, dst.Id, src.attr, e.attr, dst.attr
FROM edges AS e JOIN vertices AS src, vertices AS dst
ON e.srcId = src.Id AND e.dstId = dst.Id
```

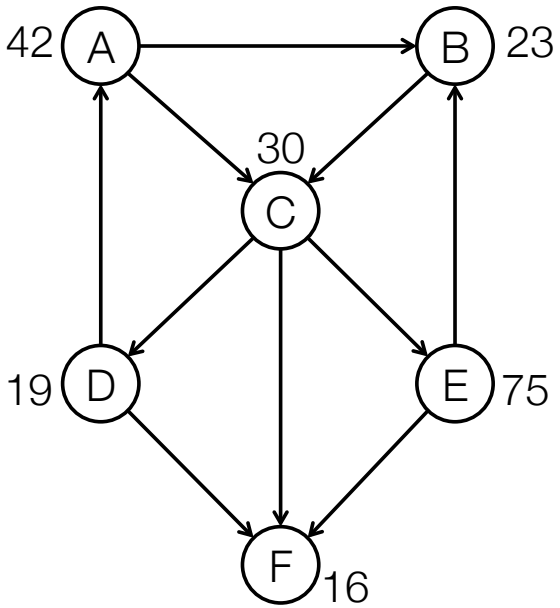


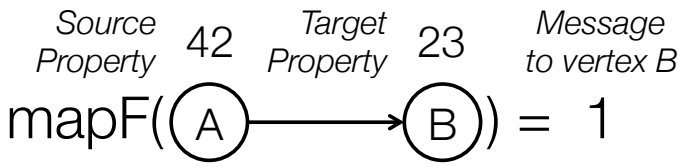
Map-Reduce for each vertex





mrTriplets (Legacy) – Oldest Ancestor Example





Resulting Vertices

Vertex Id	Property
A	0
B	2
C	1
D	1
E	0
F	3

```
val graph: Graph[User, Double]
def mapUDF(t: Triplet[User, Double]) =
  if (t.src.age > t.dst.age) 1 else 0
def reduceUDF(a: Int, b: Int): Int = a + b
val seniors: Collection[(Id, Int)] =
  graph.mrTriplets(mapUDF, reduceUDF)
```

[Gonzalez et al., 2014]

aggregateMessages Operator

`mrTriples` was replaced by `aggregateMessages` in later versions of GraphX:

- instead of sending an iterator through the `map` part, the messages are sent directly (less transmission cost)

```
class Graph[VD, ED] {
  def aggregateMessages[Msg: ClassTag](
    sendMsg: EdgeContext[VD, ED, Msg] => Unit,
    mergeMsg: (Msg, Msg) => Msg,
    tripletFields: TripletFields = TripletFields.All)
  : VertexRDD[Msg]
}
```

Previous example can be translated to:

```
val seniors = graph.aggregateMessages[Int](mapUDF, reduceUDF)
```

Pregel in GraphX

GraphX provides a Pregel operator to specify vertex programming

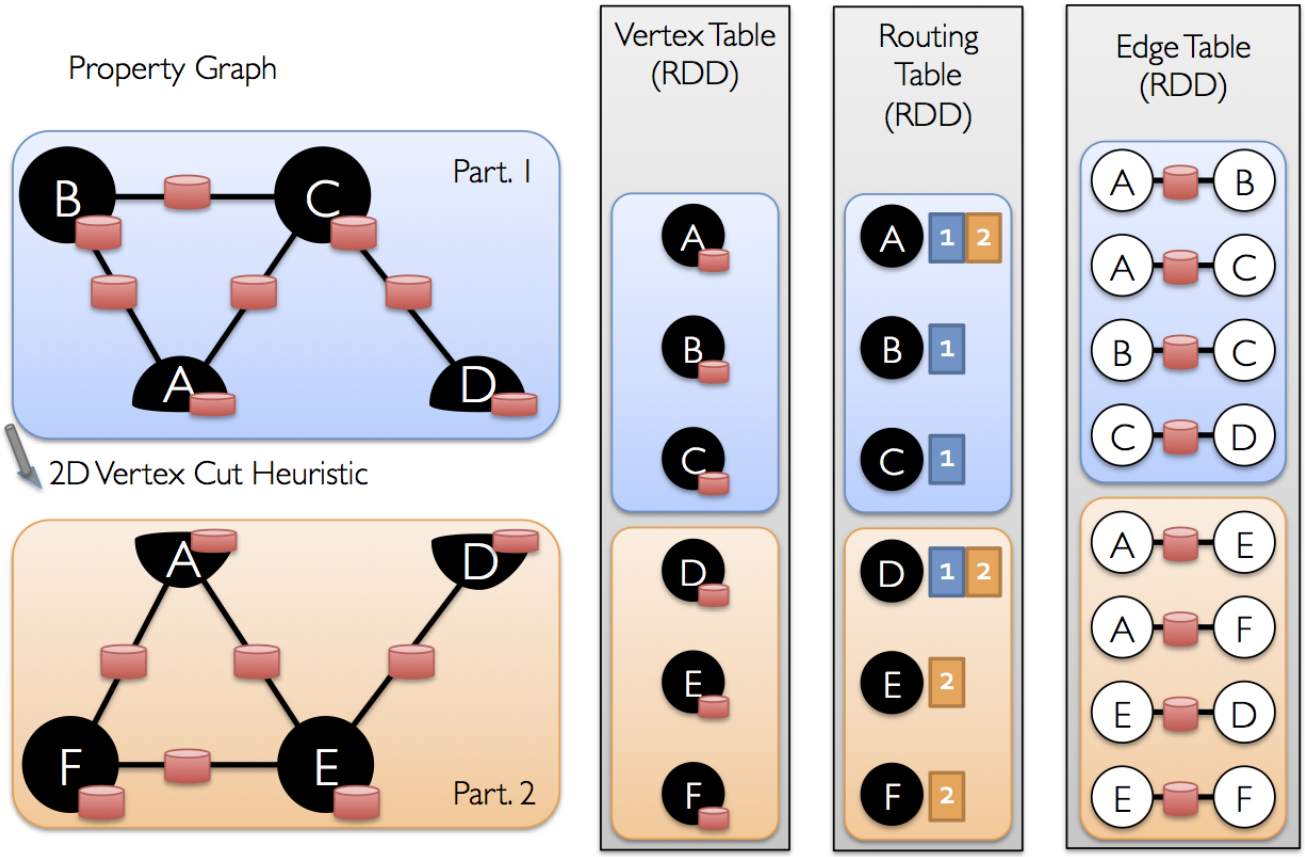
- Pregel in GraphX only sends messages to neighbors

```
val graph = GraphLoader.edgeListFile("hdfs://web.txt")
val prGraph = graph.joinVertices(graph.outDegrees)
// Implement and Run PageRank
val pageRank =
  prGraph.pregel(initialMessage = 0.0, iter = 10)(
    (oldV, msgSum) => 0.15 + 0.85 * msgSum,
    triplet => triplet.src.pr / triplet.src.deg,
    (msgA, msgB) => msgA + msgB)
// Get the top 20 pages
pageRank.vertices.top(20)(Ordering.by(_._2)).foreach(println)
```

Data Organization in GraphX

GraphX uses vertex cut partitioning

- assign edges to machines (strategy can vary), vertex attributes are on edges, routing table for neighbors on multiple machines



GraphX System Optimizations

Indexes for triplets

- useful for eliminating redundant triplets before physically scanning all triplets in `mrTriplets`

Join elimination

- if some properties are not used, then the join used to generate the triplets is rewritten to a two-way join

Further Reading

For in-depth details:

1. read the original paper on Pregel [Malewicz et al., 2010]
2. read the original paper on GraphX [Gonzalez et al., 2014]
3. read the GraphX programming guide `https://spark.apache.org/docs/latest/graphx-programming-guide.html`

Sources

The contents is partly inspired by the following lecture:

- Ankur Dave's presentation on GraphX
  `https://stanford.edu/~rezab/sparkclass/`

Some figures are also taken from the references below.

References i


Brin, S. and Page, L. (1998).
The anatomy of a large-scale hypertextual web search engine.
Comput. Networks, 30(1-7):107–117.


Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. (2012).
PowerGraph: Distributed graph-parallel computation on natural graphs.
In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, page 17–30.

References ii


Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., and Stoica, I. (2014).
GraphX: Graph processing in a distributed dataflow framework.
In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, page 599–613.


Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010).
Pregel: A system for large-scale graph processing.
In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, page 135–146.


Valiant, L. G. (1990).
A bridging model for parallel computation.
Commun. ACM, 33(8):103–111.