Seminar In Databases

Course 236826

Library For Graph Computations On Spark
Triangle Counting

For each vertex we count the number of triangles that pass through it.
Triangle Counting

For each vertex we count the number of triangles that pass through it.
Triangle Counting

For each vertex we count the number of triangles that pass through it.

The number of triangles determines the strength of a community.

The more triangles => the stronger the community
Each one starts with equal ranking (1).
Every iteration the rank of each vertex is recalculated.
Iterates until convergence.
PageRank on the Live-Journal Graph (69M Edges)

Hadoop: 1340 seconds
Spark: 354 seconds
GraphLab: 22 seconds

Hadoop is 60x slower than GraphLab
Spark is 16x slower than GraphLab
Specialized Graph Processing Systems

- Google’s Pregel
- Graph Lab
- Apache Giraph

- And many more

these systems can naturally express and efficiently execute iterative graph algorithms

Why do we need GraphX?
Specialized Graph Processing Systems

Problems:

• Not interactive

• No support for construction & post processing

• Requires maintaining multiple platforms
Multiple Views

Graphs

Tables

<table>
<thead>
<tr>
<th>Title</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multiple Views

We need separate systems to support each view:

• Data Parallel (Tables):
  - Spark
  - Hadoop

• Graph Parallel (Graphs):
  - Pregel
  - Graph Lab
  - Apache Giraph

Having separate systems for each view is:
  1. Difficult to use
  2. Inefficient
Challenges

1. Storage : How to store graphs as tables
2. Computation : How to express graph ops as table ops
3. API : How to present the two views to the user
Introducing GraphX

- **New API** that can operate on tables and graphs.
- **New Library** in Spark for graph computation- embeds Graph-Parallel model in spark

And as such:

1. Can easily integrate with RDD’s
2. Can perform Data-Parallel operations
3. Enabling the speed of specialized graph systems
The GraphX API
Property Graphs
Property Graphs

- Vertex Property:
  - User Profile
  - Current PageRank Value
Property Graphs

• Vertex Property:
  – User Profile
  – Current PageRank Value

• Edge Property:
  – Weights
  – Relationships
Example: Property Graph

Vertex Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Property(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Idan</td>
</tr>
<tr>
<td>2</td>
<td>Ofek</td>
</tr>
<tr>
<td>3</td>
<td>Shaked</td>
</tr>
</tbody>
</table>

Edge Table

<table>
<thead>
<tr>
<th>SrcId</th>
<th>DstId</th>
<th>Property(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>relative</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>friend</td>
</tr>
</tbody>
</table>

- Note: The properties can be n-tuple.
type VertexId = Long

val vertices: RDD[ (VertexId, String) ] = sc.parallelize(List(
  (1L, "Idan"),
  (2L, "Ofek"),
  (3L, "Shaked")))
Creating a Graph

type VertexId = Long

val vertices: RDD[(VertexId, String)] = sc.parallelize(List((1L, "Idan"), (2L, "Ofek"), (3L, "Shaked"))))
Creating a Graph

type VertexId = Long

val vertices: RDD[(VertexId, String)] = sc.parallelize( List(
  (1L, "Idan"),
  (2L, "Ofek"),
  (3L, "Shaked")))

class Edge[ED] ( 
  val srcld: VertexId, 
  val dstld: VertexId, 
  val attr: ED)

val edges: RDD[Edge[String]] = sc.parallelize( List(
  Edge(1L, 2L, "relative"),
  Edge(2L, 3L, "friend"))

Creating a Graph

type VertexId = Long

val vertices: RDD[(VertexId, String)] = 
  sc.parallelize(List((1L, "Idan"), (2L, "Ofek"), (3L, "Shaked")))

class Edge[ED] ( 
  val srcId: VertexId, 
  val dstId: VertexId, 
  val attr: ED)

val edges: RDD[Edge[String]] = 
  sc.parallelize(List(Edge(1L, 2L, "relative"), Edge(2L, 3L, "friend")))

val graph = Graph (vertices, edges)
class Graph[VD, ED] {
  // Table Views
  def vertices: RDD[(VertexId, VD)]
  def edges: RDD[Edge[ED]]
  def triplets: RDD[EdgeTriplet[VD, ED]]
  // Transformations
  def mapVertices[VD2](f: (VertexId, VD) => VD2): Graph[VD2, ED]
  def mapEdges[ED2](f: Edge[ED] => ED2): Graph[VD2, ED]
  def reverse: Graph[VD, ED]
  def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
                 vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
  // Joins
  def outerJoinVertices[U, VD2](tbl: RDD[(VertexId, U)]
                                )f: (VertexId, VD, Option[U]) => VD2): Graph[VD2, ED]
  // Computation
  def aggregateMessages[A](
    sendMsg: EdgeContext[VD, ED, A] 
    mergeMsg: (A, A) => A): RDD[(VertexId, A)]
class **Graph** [VD, ED] {  
    def **triplets**: RDD[ EdgeTriplet [VD, ED) ]  
}

class **EdgeTriplet** [VD, ED] (  
    val srclId: VertexId,  val dstId: VertexId,  val attr: ED  
    val srcAttr: VD , val dstAttr: VD)

<table>
<thead>
<tr>
<th>srclId</th>
<th>dstId</th>
<th>srcAttr</th>
<th>attr</th>
<th>dstAttr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Idan</td>
<td>relative</td>
<td>Ofek</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>Ofek</td>
<td>friend</td>
<td>Shaked</td>
</tr>
</tbody>
</table>
The core aggregation operation in GraphX is *aggregateMessages*.

```scala
class Graph[VD, ED] {
    def aggregateMessages[A](
        sendMsg: EdgeContext[VD, ED, A] => Unit,
        mergeMsg: (A, A) => A): RDD[(VertexId, A)]
}

class EdgeContext[VD, ED, A] (
    val srcId: VertexId, val dstId: VertexId, val attr: ED,
    val srcAttr: VD, val dstAttr: VD) {
    def sendToSrc(msg: A)
    def sendToDst(msg: A)
}

graph.aggregateMessages(
    ctx => {
        ctx.sendToSrc(1)
        ctx.sendToDst(1)
    },
    _ + _)
```
aggregateMessages

Graph

- Idan
- Ofek
- Shaked

Eli

aggregateMessages

RDD

<table>
<thead>
<tr>
<th>Vertex Id</th>
<th>degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
BEHIND THE SCENES
Graph Representation

How to store a graph as table within Spark?

Vertex Table (RDD)

Edge Table (RDD)
Graph Representation

How to store a graph as table within Spark?

Vertex Table (RDD)

Edge Table (RDD)
Graph Representation

How to store a graph as table within Spark?

Vertex Table (RDD)
- Vertex 1
- Vertex 2
- Vertex 3

Edge Table (RDD)
- Edge 1-2
- Edge 2-4
- Edge 3-5
- Edge 4-5
Graph Representation

• Because of the fact that we're storing the vertices separately from the edges:
  – We can reuse vertices or edges while changing the properties of the other.

for example:
Implementing Triplets

Vertex Table (RDD)

1
2
3
4
5

Edge Table (RDD)

1 - 2
1 - 3
2 - 3
2 - 4
3 - 5

Routing Table (RDD)

1
2
3
4
5

Routing Table (RDD)

1
2
1,2
1
2
1,2
Optimizations

GraphX adopts a vertex-cut approach to distributed graph partitioning:

- **Edge Cut**
- **Vertex Cut**
Optimizations

One example of partitioning strategy is 2D Vertex Cut Heuristic:
System Comparison

**Goal:**
- Show that GraphX, in terms of performance, is on the same level (and sometimes even faster) as specialized graph-processing systems.

Compare against Giraph, GraphLab, Spark
GraphX performs comparably to state-of-the-art graph processing systems.