Matrix and Graph Computations
Overview

Graph Computations and Pregel

Introduction to Matrix Computations
Graph Computations and Pregel
Data Flow Models

Restrict the programming interface so that the system can do more automatically

Express jobs as graphs of high-level operators
   » System picks how to split each operator into tasks and where to run each task
   » Run parts twice fault recovery

New example: Pregel (parallel graph google)
Pregel

Expose *specialized APIs* to simplify graph programming.

“Think like a vertex”
Computation depends only on the neighbors
Pregel Data Flow

- Input graph
- Vertex state 1
- Messages 1
- Superstep 1
- Vertex state 2
- Messages 2
- Superstep 2

Group by vertex ID
Simple Pregel in Spark

Separate RDDs for immutable graph state and for vertex states and messages at each iteration.

Use `groupByKey` to perform each step.

Cache the resulting vertex and message RDDs.

Optimization: co-partition input graph and vertex state RDDs to reduce communication.
Example: PageRank

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

- Rank of user \( i \)
- Weighted sum of neighbors’ ranks

Update ranks in parallel
Iterate until convergence
PageRank in Pregel

Input graph → Vertex ranks 1 → Contributions 1 → Superstep 1 (add contribs) → Vertex ranks 2 → Contributions 2 → Superstep 2 (add contribs) → ...

Group & add by vertex
GraphX

```
class Graph[VD, ED] {
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
}
```

Provides Pregel message-passing and other operators on top of RDDs
GraphX: Properties

Property Graph

Vertex Table

<table>
<thead>
<tr>
<th>Id</th>
<th>Property (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(rxin, student)</td>
</tr>
<tr>
<td>7</td>
<td>(jgonzal, postdoc)</td>
</tr>
<tr>
<td>5</td>
<td>(franklin, professor)</td>
</tr>
<tr>
<td>2</td>
<td>(istoica, professor)</td>
</tr>
</tbody>
</table>

Edge Table

<table>
<thead>
<tr>
<th>SrcId</th>
<th>DstId</th>
<th>Property (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>Collaborator</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Advisor</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>Colleague</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>PI</td>
</tr>
</tbody>
</table>
GraphX: Triplets

The *triplets* operator joins vertices and edges:

- Vertices: A
- Edges: A → B
- Triplets: A → B

![Diagram](image-url)
Map Reduce Triplets

Map-Reduce for each vertex:

\[
\text{mapF}(A \rightarrow B) \rightarrow A_1 \\
\text{mapF}(A \rightarrow C) \rightarrow A_2 \\
\text{reduceF}(A_1, A_2) \rightarrow A
\]
Example: Oldest Follower

What is the age of the oldest follower for each user?

```scala
val oldestFollowerAge = graph
  .mrTriplets(
    e => (e.dst.id, e.src.age), //Map
    (a, b) => max(a, b) //Reduce
  )
  .vertices
```
Summary of Operators

All operations:

https://spark.apache.org/docs/latest/graphx-programming-guide.html#summary-list-of-operators

Pregel API:

https://spark.apache.org/docs/latest/graphx-programming-guide.html#pregel-api
The GraphX Stack
(Lines of Code)

PageRank (5)
Connected Comp. (10)
Shortest Path (10)
SVD (40)
ALS (40)
K-core (51)
Triangle Count (45)
LDA (120)

Pregel (28) + GraphLab (50)

GraphX (3575)

Spark
Optimizations

Overloaded vertices have their work distributed

Edge Cut

Vertex Cut
Optimizations

Property Graph

Vertex Table (RDD)

Routing Table (RDD)

Edge Table (RDD)

2D Vertex Cut Heuristic
More examples

In your HW: Single-Source-Shortest Paths using Pregel
Distributing Matrix Computations
Distributing Matrices

How to distribute a matrix across machines?
» By Entries (CoordinateMatrix)
» By Rows (RowMatrix)
» By Blocks (BlockMatrix)

All of Linear Algebra to be rebuilt using these partitioning schemes

As of version 1.3
Distributing Matrices

Even the simplest operations require thinking about communication e.g. multiplication

How many different matrix multiplies needed?

» At least one per pair of \{Coordinate, Row, Block, LocalDense, LocalSparse\} = 10

» More because multiplies not commutative
Distributed Singular Value Decomposition
Singular Value Decomposition

\[ A_{m \times n} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \]
Singular Value Decomposition

Two cases

» Tall and Skinny

» Short and Fat (not really)

» Roughly Square

SVD method on RowMatrix takes care of which one to call.
Tall and Skinny SVD

- Given $m \times n$ matrix $A$, with $m \gg n$.
- We compute $A^T A$.
- $A^T A$ is $n \times n$, considerably smaller than $A$.
- $A^T A$ is dense.
- Holds dot products between all pairs of columns of $A$.

\[
A = U \Sigma V^T \quad \quad \quad \quad A^T A = V \Sigma^2 V^T
\]
Tall and Skinny SVD

\[ A^T A = V \Sigma^2 V^T \]

Gets us \( V \) and the singular values

\[ A = U \Sigma V^T \]

Gets us \( U \) by one matrix multiplication
Square SVD

ARPACK: Very mature Fortran77 package for computing eigenvalue decompositions

JNI interface available via netlib-java

Distributed using Spark – how?
Square SVD via ARPACK

Only interfaces with distributed matrix via matrix-vector multiplies

\[ K_n = [b \quad Ab \quad A^2b \quad \cdots \quad A^{n-1}b] \]

The result of matrix-vector multiply is small.

The multiplication can be distributed.
Square SVD

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Number of nonzeros</th>
<th>Time per iteration (s)</th>
<th>Total time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$23,000,000 \times 38,000$</td>
<td>$51,000,000$</td>
<td>0.2</td>
<td>10</td>
</tr>
<tr>
<td>$63,000,000 \times 49,000$</td>
<td>$440,000,000$</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>$94,000,000 \times 4,000$</td>
<td>$1,600,000,000$</td>
<td>0.5</td>
<td>50</td>
</tr>
</tbody>
</table>

With 68 executors and 8GB memory in each, looking for the top 5 singular vectors