Secure RDD for SparkFHE

Sai Sujith Kammari
Dr. Peizhao Hu (Advisor)
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Project Refresh: What is Secure RDD</td>
</tr>
<tr>
<td>02</td>
<td>RDD: What exactly is Resilient Distributed Dataset (RDD)</td>
</tr>
<tr>
<td>03</td>
<td>Data Types in RDD: What are different data types based on RDD in Spark MLlib</td>
</tr>
<tr>
<td>04</td>
<td>Local Data Types: What data types exist to store data on single machine</td>
</tr>
<tr>
<td>05</td>
<td>Distributed Data Types: What data types exist to distribute data in multiple machines</td>
</tr>
<tr>
<td>06</td>
<td>Summary: Goal, progress and next steps</td>
</tr>
</tbody>
</table>
What is Secure RDD for SparkFHE
Secure RDD for SparkFHE

- Apache Spark is the most popular open-source, distributed processing system used for big data workloads.
- Workloads in Apache Spark:
  a. Core
  b. MLlib
  c. Streaming
  d. SQL
  e. GraphX
- Apache Spark provides high-level APIs in Java, Scala, Python and R, and supports code reuse across multiple workloads.
- Implementing homomorphic encryption in the RDDs which are used in MLlib.
Resilient Distributed Dataset

What is a Resilient Distributed Dataset
Resilient Distributed Datasets

- RDD is a fundamental data structure of Spark which can contain any type of Python, Java or Scala objects.
- Dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.
- RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- RDD can be created in two ways:
  a. Parallelizing and existing collection.
  b. Referencing external supported datasets like text file, SequenceFiles and any other Hadoop InputFormat.
- Elements of the collection are copied to form a distributed dataset.
- Once created, all dataset operations can be acted on these RDDs.
Data types in RDD

What are different data types based on RDD in Spark MLlib
Data Types in Spark MLlib

- **Local Vector**
  - Dense Vector
  - Sparse Vector

- **Labeled Point**

- **Local Matrix**
  - Dense Matrix
  - Sparse Matrix
  - Indexed Row Matrix
  - Coordinate Matrix
  - Block Matrix

- **Distributed Matrix**
Local Data Types

What data types exist to store data on single machine
A local vector has integer-typed indices and double-typed values.

Two types:
- Dense vector
- Sparse vector
A Labeled Point is a local vector, either dense or sparse, associated with a label/response.

Double value is stored as a label, so this data structure can be used in both regression and classification.
Local Matrix

A local matrix has integer-typed rows and column indices and double-typed values, stored on a single machine.

Two types:
- Dense Matrix
- Sparse Matrix

Matrices.dense(3, 2, Array(1.0, 0.0, 5.0; 2.0, 4.0, 0.0))

Matrices.sparse(4, 5,
Array(0, 2, 3, 6, 7, 8),
Array(0, 3, 1, 0, 2, 3, 2, 1),
Array(1.0, 14.0, 6.0, 2.0, 11.0, 16.0, 12.0, 9.0))

![Local Matrix Diagram](https://medium.com/@rickynguyen/getting-started-with-spark-day-5-36b62a6d13bf)
Distributed Data types

What data types exist to distribute data in multiple machines
A Row matrix is a row-oriented distributed matrix with NO meaningful row indices.

Row Matrix is just a collection of feature vectors.

Each row in RDD is a local vector.

Max number of columns is limited by the integer range.
Indexed Row Matrix

An Indexed Row Matrix is similar to a row matrix but **WITH** meaningful row indices backed by RDD of indexed rows.

Each row is represented by long-typed index and a local vector.
Distributed Matrix

The values are explicitly defined by using IndexedRow(row_index, col_index, value)

Should be used only when both dimensions of the matrix are huge, and the matrix is very sparse.

Coordinate Matrix

The values are explicitly defined by using IndexedRow(row_index, col_index, value)

Should be used only when both dimensions of the matrix are huge, and the matrix is very sparse.
Distributed Matrix

- Used to store different sub-matrices of a large matrix on different machines
- Supports add and multiply operations with another block matrix

Block Matrix

- Used to store different sub-matrices of a large matrix on different machines
- Supports add and multiply operations with another block matrix
06

Summary

Goal, progress and next steps
Summary

Goal
Implement homomorphic encryption in MLlib module of Apache Spark by modifying the underlying data types

Progress
Understand working of underlying data structures of MLlib. Check the feasibility of implementation.

Next Steps
Start implementing homomorphic encryption in the data types and build an example to test/demo the implementation.
Any Questions
THANKS

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.

Please keep this slide for attribution.