Advanced Analytics With Splunk Using Apache Spark Machine Learning And Spark Graph

Raanan Dagan  |  Architect

September 25, 2017  |  Washington, DC
Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2017 Splunk Inc. All rights reserved.
Why Spark?

- Most of machine learning algorithms are iterative because each iteration can improve the results
- With disk based approach each iteration’s output is written to disk, making it slow

Hadoop execution flow

Spark execution flow

http://www.wiziq.com/blog/hype-around-apache-spark/
About Apache Spark

- Initially started at UC Berkeley in 2009
- Fast and general purpose cluster computing system
- 10x (on disk) - 100x (In-Memory) faster
- Most popular for running *Iterative Machine Learning Algorithms.*
- Provides high level APIs in
  - Java, Scala, Python
- Integration with Hadoop and its ecosystem and can read existing data

http://spark.apache.org/
Spark Core
Spark Core

- Spark Core contains the basic functionality of Spark
  - Task scheduling
  - Memory management
  - Fault recovery
  - Interacting with storage systems

- Home to **Resilient Distributed Datasets (RDDs)**
- Provides many APIs for building and manipulating RDD
Resilient Distributed Dataset (RDD)

- Resilient Distributed Dataset (RDD) is a basic abstraction in Spark
- Immutable, partitioned collection of elements that can be operated in parallel
- Basic Operations
  - map
  - filter
  - persist
- Multiple Implementation
  - PairRDDFunctions: RDD of Key-Value Pairs, groupByKey, Join
- RDD main characteristics:
  - A list of partitions
  - A function for computing each split
Spark SQL
DataFrames
Interfaces to Spark SQL
Most powerful way to use Spark SQL is inside a Spark application

Load data and query it with SQL while simultaneously combining it with “regular” program code utilizing SQLContext or HiveContext

```scala
// SQL Imports
// Import Spark SQL. If you can't have the
// hive dependencies
import org.apache.spark.sql.SQLContext

// Construct SQL Context
val sqlContext = new SQLContext(...)  

// SQL Imports
// Import Spark SQL
import org.apache.spark.sql.hive.HiveContext

// Construct Hive Context
val hiveContext = new HiveContext(...)  
```
HiveContext (Recommended)

- Provides a superset of the functionality in addition to the basic SQLContext
- Write queries using the more complete HiveQL parser
- Access to Hive UDFs and ability to read data from Hive tables
- Build DataFrames (represent structure data), and operate on them with SQL or with normal RDD operations like map
DataFrames

▶️ Offers rich relational/procedural integration within Spark programs

▶️ DataFrames:
  • Collections of structured records that can be manipulated using Spark’s procedural API or new relational API
  • Perform relational operations on DataFrames using a domain-specific language (DSL) similar to R data frames and Python Pandas
  • Pass Scala, Java or Python functions through DataFrames to build a logical plan
  • Create directly from Spark’s distributed objects
  • Enable relational relational processing in existing Spark programs

▶️ Automatically store data in a columnar format
▶️ Go through a relational optimizer, Catalyst
▶️ Standard data representation in a new “ML pipeline” API for machine learning
Query Federation To External Databases

- Data pipelines often combine data from heterogeneous sources
- Spark SQL data sources leverage Catalyst to push predicates down into the data sources whenever possible

Example: Use JDBC data source and JSON data source to join two tables together

```sql
CREATE TEMPORARY TABLE users USING jdbc
OPTIONS(driver "mysql" url "jdbc:mysql://userDB/users ")
CREATE TEMPORARY TABLE logs
USING json OPTIONS (path "logs.json")
SELECT users.id,users.name,logs.message
FROM users JOIN logs WHERE users.id=logs.userId
AND users.registrationDate > "2015-01-01"
```
Spark MLlib
ML algorithms include:
- Classification: logistic regression, naive Bayes, ...
- Regression: generalized linear regression, survival regression...
- Decision trees, random forests, and gradient-boosted trees
- Recommendation: alternating least squares (ALS)
- Clustering: K-means, Gaussian mixtures (GMMs), ...
- Topic modeling: latent Dirichlet allocation (LDA)
- Frequent itemsets, association rules, and sequential pattern mining

ML workflow utilities include:
- Feature transformations: standardization, normalization, hashing, ...
- ML Pipeline construction
- Model evaluation and hyper-parameter tuning
- ML persistence: saving and loading models and Pipelines
- Distributed linear algebra: SVD, PCA, ...
- Statistics: summary statistics, hypothesis testing, ...

Spark Machine Learning Basics
Spark Classification ML Example

Supervised learning for predicting discrete labels

Multiple algorithms

- logistic regression
- Decision tree classifier
- Random forest classifier
- Gradient boosted tree classifier
- Multi-layer neural network classifier
Spark Classification ML Code Example

1. Extract Fields

```scala
val trainingDataTable = sql(""
SELECT e.action
    , u.age,
    , u.latitude,
    , u.longitude
FROM Users u
JOIN Events e
ON u.userId = e.userId"")
```

2. Build Model

```scala
val trainingData = trainingDataTable.map { row =>
  val model =
    new LogisticRegressionWithSGD().run(trainingData)
}
```

3. Predict

```scala
case class Score(userId: Int, score: Double)
val scores = allCandidates.map { row =>
  val features = Array[Double](row(1), row(2), row(3))
  Score(row(0), model.predict(features))
}
```
Spark GraphX

Multiple Algorithms

- PageRank
- Connected components
- Label propagation
- SVD++
- Strongly connected components
- Triangle count

Vertex Property:
- User Profile
- Current PageRank Value

Edge Property:
- Weights
- Relationships
- Timestamps
Spark GraphX Example

### Property Graph

```
<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>
```

### Vertex 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>
```

```scala
// Assume the SparkContext has already been constructed
val sc: SparkContext

// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
                      (5L, ("franklin", "professor")), (2L, ("istoica", "professor"))))

// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(3L, 7L, "collab"), Edge(5L, 3L, "advisor"),
                      Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))

// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")

// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)
```
Spark GraphX Architecture

Property Graph

Vertex Act

Part. 1

Vertex Table (RDD)

Routing Table (RDD)

Edge Table (RDD)

Part. 2
Spark Stream

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches
<table>
<thead>
<tr>
<th></th>
<th>Kafka Streams</th>
<th>Storm</th>
<th>Spark Streaming</th>
<th>Flink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration</td>
<td>Easy</td>
<td>Difficult</td>
<td>Difficult</td>
<td>Difficult</td>
</tr>
<tr>
<td>Development</td>
<td>Easy, flexible</td>
<td>Difficult</td>
<td>Difficult</td>
<td>Difficult</td>
</tr>
<tr>
<td>Operations</td>
<td>Easy</td>
<td>Difficult (Clustering)</td>
<td>Difficult (Clustering)</td>
<td>Difficult (Clustering)</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Small</td>
<td>Large (Clustering)</td>
<td>Large (Clustering)</td>
<td>Large (Clustering)</td>
</tr>
<tr>
<td>Delivery</td>
<td>At least once</td>
<td>At least once</td>
<td>Exactly Once</td>
<td>Exactly Once</td>
</tr>
<tr>
<td>Latency</td>
<td>Milliseconds</td>
<td>Seconds</td>
<td>Milliseconds</td>
<td>Milliseconds</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Scalability</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Document Classification With Splunk And Spark
2016 Spark Survey

**Types of Products Built**

- 68% Business / Customer Intelligence
- 52% Data Warehousing
- 45% Real-Time / Streaming Solutions
- 40% Recommendation Engines
- 37% Log Processing
- 36% User-Facing Services
- 29% Fraud Detection / Security

**Languages Used in Spark Year-Over-Year**

- SQL: 36% (2015), 44% (2016)
- R: 18% (2015), 20% (2016)
- Python: 58% (2015), 62% (2016)
- Scala: 71% (2015), 65% (2016)
- Java: 31% (2015), 29% (2016)

**Spark Components Used in Production Year-Over-Year**

- DataFrames: 15% (2015), 38% (2016)
- SQL: 24% (2015), 40% (2016)
- Streaming: 14% (2015), 22% (2016)
- Advanced Analytics (MLlib): 13% (2015), 18% (2016)
Document Classification: Why Spark?

Problem: Spark processing does not provide easy analytics or any visualizations

Goal: Allow analysts and regulators the ability to know exactly where each file exists in the system

Solution: Apache Nifi collect all new files from NFS and stores it on Hadoop. Spark Core, Spark Machine Learning, and Apache Tika create Metadata classification. Splunk Analytics for Hadoop exposes metadata classification files to end users.
Spark SQL And Splunk
Spark SQL And Splunk

db_connection_types.conf

[spark_sql]
displayName = Spark SQL
serviceClass = com.splunk.dbx2.sparksql.SparkSqlJDBC
jdbcUrlFormat = jdbc:spark://<Thrift Server Host>:<Thrift Server Port>/<database>
jdbcDriverClass = com.simba.spark.jdbc41.Driver
Spark SQL And Splunk

```sql
| dbxquery query="SELECT * FROM `Spark`.`xademo`.`customer_details`" connection="spark_local_2" wrap=t
```

30 results (before 8/22/16 10:34:23.000 PM) No Event Sampling

<table>
<thead>
<tr>
<th>Events</th>
<th>Patterns</th>
<th>Statistics (30)</th>
<th>Visualization</th>
</tr>
</thead>
</table>

| 20 Per Page | Format | Preview |

<table>
<thead>
<tr>
<th>(001) customer_details.phone_number</th>
<th>(002) customer_details.plan</th>
<th>(003) customer_details.rec_date</th>
<th>(004) customer_details.status</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHONE_NUM</td>
<td>PLAN</td>
<td>REC_DATE</td>
<td>STATUS</td>
</tr>
</tbody>
</table>
Spark ML → Splunk

Spark SQL with Spark Mllib:
Thank You

Don't forget to rate this session in the .conf2017 mobile app