Key Takeaways

- Learn about graph data processing and analytics
- Apache Spark GraphX library as a solution to perform graph data analytics
- Graph algorithms like PageRank, Connected Components and Triangle Counting
- Spark GraphX components and API
- Sample application using Spark GraphX

This is the sixth article of the "Big Data Processing with Apache Spark" series. Please see also: Part 1: Introduction, Part 2: Spark SQL, Part 3: Spark Streaming, Part 4: Spark Machine Learning,Part 5: Spark ML Data Pipelines.

Big data comes in different shapes and sizes. It can be batch data that needs to be processed offline, processing large set of records and generating the results and insights at a later time. Or the data can be real-time streams which needs to be processed on the fly and create the data insights almost instantaneously.

We have seen how Apache Spark can be used for processing batch (Spark Core) as well as real-time data (Spark Streaming).

Sometimes the data we need to deal with is connected in nature. For example, in a social media application, we have entities like Users, Articles, Likes etc. that need to be managed and processed as a single logical unit of data. This type of data is called Graph data, and requires a different type of techniques and approaches to run analytics on this data, compared to traditional data processing.

In the previous articles in this article seriestitled “Big Data Processing with Apache Spark”, we learned about the Apache Spark framework and its different libraries for big data processing starting with the first article on Spark Introduction (Part 1), then we looked at the specific libraries like Spark SQL library (Part 2), Spark Streaming (Part 3), and both Machine Learning packages: Spark MLlib (Part 4) and Spark ML (Part 5).

In this final installment, we will focus on how to process graph data and Spark’s graph data analytics library called GraphX.

Related Vendor Content

NoSQL Technical Comparison Report

Building Scalable Web Applications with Azure Database for MySQL

Docker and a Native Linux Experience: What's New in SQL Server 2017

Designing & Building Architecture for Microservices (By O’Reilly)

AWS Monitoring: Visualize the Health of Your Infrastructure Components Instantly

Related Sponsor

This guide walks you through your first NoSQL project, from best use case through examples of code.

First, let's look at what graph data is and why it's critical to process this type of data in enterprise big data applications.

Graph Data

There are three different topics to cover when we discuss graph data related technologies:

- Graph Databases
- Graph Data Analytics
- Graph Data Visualization

Let's discuss these topics briefly to learn how they are different from each other and how they complement each other to help us develop a comprehensive graph based big data processing and analytics architecture.

Graph Databases

Unlike traditional data models, data entities as well as the relationships between those entities are the core elements in graph data models. When working on graph data, we are interested in the entities and the connections between the entities.

For example, if we are working on a social network application, we would be interested in the details of a particular user (let's say John) but we would also want to model, store and retrieve the associations between this user and other users in the network. Examples of these associations are "John is a friend of Mike" or "John read the book authored by Bob."

It's important to remember that the graph data we use in the real world applications is dynamic in nature and changes over time.

The advantage of graph databases is to uncover patterns that are usually difficult to detect using traditional data models and analytics approaches.

Without Graph databases, implementing a use case like finding common friends is an expensive query as described in this post using data from all the tables with complex joins and query criteria.

Graph database examples include Neo4j, DataStax Enterprise Graph, AllegroGraph, InfiniteGraph, and OrientDB.

Graph Data Modeling

Graph data modeling effort includes defining the nodes (also known as vertices), relationships (also known as edges), and labels to those nodes and relationships.

Graph databases are modeled based on what Jim Webber calls Query-driven Modeling which means the data model is open to domain experts rather than just database specialists and supports team collaboration for modeling and evolution.

Graph database products typically include a query language (Cypher if you are using Neo4j as the database) to manage the graph data stored in the database.

Graph Data Processing

Graph data processing mainly includes graph traversal to find specific nodes in the graph data set that match the specified patterns and then locate the associated nodes and relationships in the data so we can see the patterns of connections between different entities.

The data processing pipeline typically includes the following steps:

- pre-processing of data (which includes loading, transformation, and filtering)
- graph creation
- analysis
- post-processing

A typical graph analytics tool should provide the flexibility to work with both graphs and collections so we can combine data analytics tasks like ETL, exploratory analysis, and iterative graph computation within a single system without having to use several different frameworks and tools.

There are several frameworks that we can use for processing graph data and running predictive analytics on the data. These frameworks include Spark GraphX, Apache Flink's Gelly, and GraphLab.

In this article, we'll focus on Spark GraphX for analyzing the graph data.

There are also several different graph generators as discussed in Gelly framework documentation like Cycle Graph, Grid Graph, Hypercube Graph, Path Graph and Star Graph.

Graph Data Visualization

Once we start storing connected data in a graph database and run analytics on the graph data, we need tools to visualize the patterns behind the relationships between the data entities.

Graph data visualization tools include D3.js, Linkurious and GraphLab Canvas. Data analytics efforts are not complete without data visualization tools.

Graph Use Cases

There are a variety of use cases where graph databases are better fit to manage the data than other solutions like relational databases or other NoSQL data stores. Some of these use cases include the following:

- **Recommendations and Personalization:** Graph analysis can be used to generate recommendation and personalization models on their customers and to make key decisions from the insights found in the data analysis. This helps the enterprises to effectively influence customers to purchase their product. This analysis also helps with marketing strategy and customer service behavior.
- **Fraud Detection:** Graph data solutions also help to find fraudulent transactions in a payment processing application based on the connected data that include the entities like users, products, transactions, and events. This article describes a test application about how to use Spark GraphX for fraud detection using PageRank algorithm on metadata about phone communication.
- **Topic Modeling:** This includes techniques to cluster documents and extract topical representations from the data in those documents.
- **Community Detection:** Alibaba website uses graph data analytics techniques like community detection to solve ecommerce problems.
- **Flight Performance:** Other use cases include on-time flight performance as discussed in this article, to analyze flight performance data organized in graph structures and find out statistics like airport ranking and shortest paths between cities.
- **Shortest Distance:** Shortest distances and paths are also useful in social network applications. They can be used for measuring the relevance of a particular user in the network. Users with smaller shortest distances are more relevant than users farther away.

Spark GraphX

GraphX is Apache Spark's API for graphs and graph-parallel computation. It extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge.

GraphX library provides graph operators like `subgraph`, `joinVertices`, and `aggregateMessages` to transform the graph data. It provides several ways of building a graph from a collection of vertices and edges in an RDD or on disk. GraphX also includes a number of graph algorithms and builders to perform graph analytics tasks. We'll discuss graph algorithms later in this article.

Figure 1 below shows Apache Spark ecosystem and where GraphX fits in with other libraries in the framework.

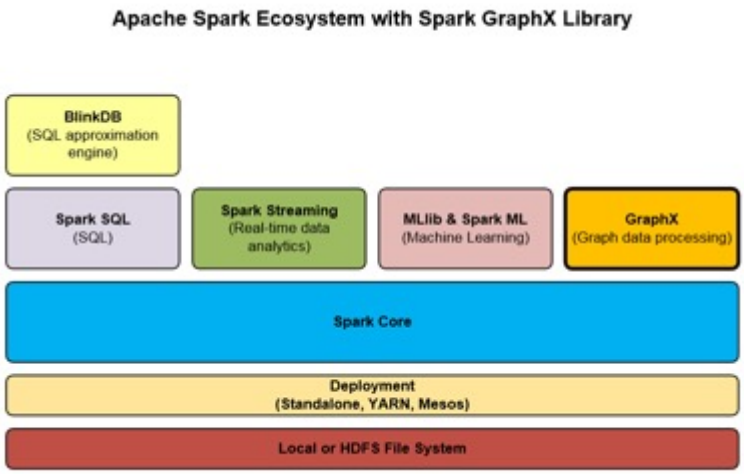

Figure 1. Spark Ecosystem and GraphX library

GraphX makes it easier to run analytics on graph data with the built-in operators and algorithms. It also allows us to cache and uncache the graph data to avoid recomputation when we need to call a graph multiple times.

Some of the graph operators available in GraphX are listed in Table 1 below.

Operator Type	Operators	Description
Basic Operators	• numEdges • numVertices • inDegrees • outDegrees • degrees	
Property Operators	• mapVertices • mapEdges • mapTriplets	
Structural Operators	• reverse • subgraph • mask • groupEdges	
Join Operators	• joinVertices • outerJoinVertices	

Table 1: Spark GraphX's graph operators

We'll look at more details of these operators in the Sample Application section when we run GraphX algorithms on different social network data sets.

GraphFrames

GraphFrames, a new addition to Spark graph data processing toolset, integrates the features like pattern matching and graph algorithms with Spark SQL. Vertices and edges are represented as DataFrames instead of RDD objects.

GraphFrames simplify the graph data analytics pipeline and optimize the queries across both graph and relational data. It provides some advantages as shown below compared to the RDD based graph data processing:

- Support for Python and Java in addition to Scala APIs. Now we can use GraphX algorithms in all three languages.
- Advanced query capability using the Spark SQL and DataFrames API. Graph-aware query planner uses materialized views to improve the query performance.
- We can also save and load graphs using formats like Parquet, JSON, and CSV.

GraphFrames is available as an add-on component to GraphX from spark-apache.org website. This article shows how to use GraphFrames to calculate the PageRank for each node in the graph data set.

Graph Analytics Algorithms

Graph algorithms help with executing the analytics on graph data sets without having to write our own implementations of those algorithms. Below is a list of various algorithms that help with finding patterns in the graphs.

- PageRank
- Connected components
- Label propagation
- SVD++
- Strongly connected components
- Triangle count
- Single-Source-Shortest-Paths
- Community Detection

Spark GraphX comes with a set of pre-built graph algorithms to help with graph data processing and analytics tasks. These algorithms are available in the org.apache.spark.graphx.lib package. It's as simple as calling these algorithms as methods in Graph class.

Figure 2 below shows how the different graph algorithms are built on top of the base GraphX API.

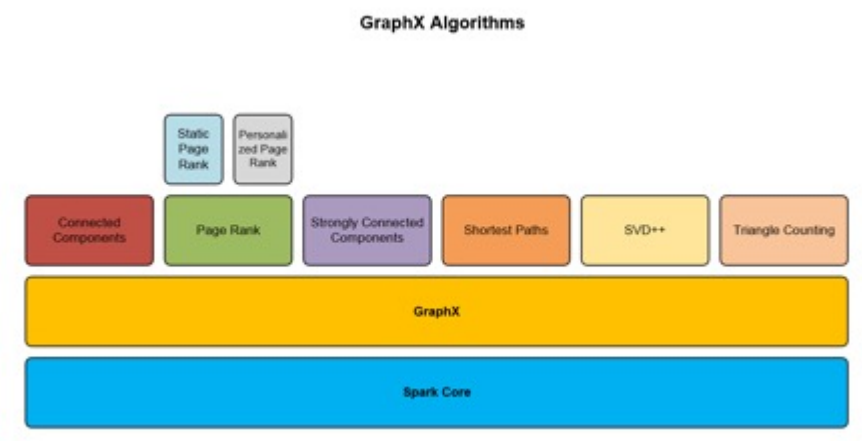

Figure 2. Graph algorithms in Spark GraphX library

In this article, we'll look into more details of the PageRank, Connected Components, and Triangle Count algorithms.

PageRank

PageRank algorithm is used to determine the relative importance of an object inside a graph data set. It measures the importance of each node in a graph, assuming an edge from another node to this node represents an endorsement.

Google's search engine is a classic example of PageRank. Google uses PageRank as one of the measures to determine the importance of a web page based on the how many other web pages reference it.

Another example is social network website like Twitter. If a Twitter user is followed by lot of other users, then that user has a higher influence in the network. This metric can be used for ad selection/placement to the users that follow the first user (100,000 users follow a chef=> probably food lovers)

GraphX provides two implementations of PageRank: Static and Dynamic.

Static PageRank: This algorithm runs for a fixed number of iterations to generate PageRank values for a given set of nodes in a graph data set.

Dynamic PageRank: On the other hand, Dynamic PageRank algorithm runs until PageRank values converge based on a pre-defined tolerance value.

Connected Components

A Connected Component in a graph is a connected subgraph where two vertices are connected to each other by an edge and there are no additional vertices in the main graph. This means the two nodes belong to the same connected component when there is a relationship between them. The lowest numbered vertex number of ID in the subgraph is used to label the connected components in a graph. Connected components can be used to create clusters in the graph for example in a social network.

There are two ways of traversing the graph for computing connected components:

- Breadth-first Search (BFS)
- Depth-first Search (DFS)

There is another algorithm called Strongly Connected Components (SCC) in graph data processing. If all nodes in a graph are reachable from every single node, then the graph is considered to be strongly connected.

Triangle Counting

Triangle counting is a community detection graph algorithm which is used to determine the number of triangles passing through each vertex in the graph data set. A vertex is part of a triangle when it has two adjacent vertices with an edge between. The triangle is a three-node subgraph, where every two nodes are connected. This algorithm returns a Graph object and we extract vertices from this triangle counting graph.

Triangle counting is used heavily in social network analysis. It provides a measure of clustering in the graph data which is useful for finding communities and measuring the cohesiveness of local communities in social network websites like LinkedIn or Facebook. Clustering Coefficient, an important metric in a social network, shows how much community around one node is tightly connected.

Other use cases where Triangle Counting algorithm is used are spam detection and link recommendations.

Triangle counting is a message heavy and computationally expensive algorithm compared to other graph algorithms. So, make sure you run the Spark program on a decent computer when you test Triangle Count algorithm. Note that PageRank is a measure of relevancy whereas Triangle Count is a measure of clustering.

Sample Application

We have seen so far in this article what graph data is and why graph analytics is an important part of data processing projects in different organizations. Let's now look at a sample application that uses some of the graph algorithms.

We'll use data sets from different social network websites like Facebook, LiveJournal, and YouTube. All these applications contain the connected data and are excellent resources for graph data analytics programs.

The examples we use in this article are based on the GraphX samples discussed in this article on comparison of graph processing tools.

Use Case

The main goal of the use cases in our sample application is to determine graph data statistics such as:

- How popular different users in the social network are (PageRank)
- Clusters of users based on how the users in the network are connected (Connected Components)
- Community detection and cohesiveness of the communities of users in the social network (Triangle Counting)

Datasets

In our code examples on Spark GraphX, we will use few different data sets for running Spark GraphX programs. These datasets are available from SNAP (Stanford Network Analysis Project

(SNAP) website hosted by Stanford University. If you want to download these datasets, copy them to data folder in the sample application main directory.

Algorithm

We'll use the following three algorithms in our sample application:

- PageRank on YouTube
- Connected Components on LiveJournal
- Triangle Counting on Facebook

The following table shows the use cases, data sets, and algorithms used in the graph data processing programs.

Use Case	Dataset Source	Link	File Name	Rename File
PageRank	YouTube	https://snap.stanford.edu/data/com-Youtube.html	com-youtube.ungraph.txt	page-rank-yt-data.txt
Connected Components	LiveJournal	https://snap.stanford.edu/data/com-LiveJournal.html	com-lj.ungraph.txt	connected-components-lj-data.txt
Triangle Count	Facebook	https://snap.stanford.edu/data/egonets-Facebook.html	facebook_combined.txt	triangle-count-fb-data.txt

Table 2: Data sets and algorithms used in Spark GraphX sample application

Once you rename the files, copy them to a subdirectory called "data" in the project's main directory.

Technologies

We'll use the following technologies in graph analytics sample code:

Technology	Version
Apache Spark	2.1.0
Scala	2.11
JDK	1.8
Maven	3.3

Table 3. Technologies and tools used in sample application.

Example Code

We'll write Spark GraphX code using Scala programming language. We'll use Spark Shellcommand line tool to run these programs. This is the fastest way to verify the results of the program. No additional code compilation and build steps are needed.

Before we look at the specific code for each of the use cases, these programs will be available as a zip file along with this article that you can download and try out in your own development environment.

Let's look at the details of each of the sample GraphX programs.

First, we will run PageRank on YouTube online social network data. This dataset includes the ground-truth communities which are basically user defined groups that other users can join in.

PageRank:

```
import org.apache.spark._
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
import java.util.Calendar

// Load the edges first
val graph = GraphLoader.edgeListFile(sc, "data/page-rank-yt-data.txt")

// Compute the graph details like edges, vertices etc.

val vertexCount = graph.numVertices

val vertices = graph.vertices
vertices.count()

val edgeCount = graph.numEdges

val edges = graph.edges
edges.count()

//
// Let's look at some of the Spark GraphX API like triplets, indegrees, and out
//
val triplets = graph.triplets
triplets.count()
triplets.take(5)

val inDegrees = graph.inDegrees
inDegrees.collect()

val outDegrees = graph.outDegrees
outDegrees.collect()

val degrees = graph.degrees
degrees.collect()

// Number of iterations as the argument
```

```
val staticPageRank = graph.staticPageRank(10)
staticPageRank.vertices.collect()

Calendar.getInstance().getTime()
val pageRank = graph.pageRank(0.001).vertices
Calendar.getInstance().getTime()

// Print top 5 items from the result
println(pageRank.top(5).mkString("\n"))
```

The variable “sc” in the above code is the SparkContext which is already available when you run programs from Spark Shell.

Let’s now look at the code for how to run Connected Components on LiveJournal’s social network data. This dataset includes the users who are registered on the website and maintain individual and group blog posts. The website also allows users to identify other users who are their friends.

Connected Components:

```
import org.apache.spark._
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
import java.util.Calendar

// Connected Components
val graph = GraphLoader.edgeListFile(sc, "data/connected-components-lj-data.txt

Calendar.getInstance().getTime()
val cc = graph.connectedComponents()
Calendar.getInstance().getTime()

cc.vertices.collect()

// Print top 5 items from the result
println(cc.vertices.take(5).mkString("\n"))

val scc = graph.stronglyConnectedComponents()
scc.vertices.collect()
```

Finally, here is the Spark program, again in Scala, for calculating Triangle Counting on Facebook's social circles data. The dataset includes the lists of friends from Facebook with user profiles, circles, and ego networks.

Triangle Counting:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
```

```
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

val graph = GraphLoader.edgeListFile(sc,"data/triangle-count-fb-data.txt")

println("Number of vertices : " + graph.vertices.count())
println("Number of edges : " + graph.edges.count())

graph.vertices.foreach(v => println(v))

val tc = graph.triangleCount()

tc.vertices.collect

println("tc: " + tc.vertices.take(5).mkString("\n"));

// println("Triangle counts: " + graph.connectedComponents.triangleCount().vert

println("Triangle counts: " + graph.connectedComponents.triangleCount().vertice

val sum = tc.vertices.map(a => a._2).reduce((a, b) => a + b)
```

Conclusions

With the increasing growth of connected data in commercial organizations, government agencies, and social media networking companies, graph data processing and analytics are only going to become more critical in predictive analytics and recommendation engine solutions to gain insights and provide service for employees, customers and users.

As we learned in this article, Spark GraphX is a very good choice for graph data processing requirements. It provides a unified data processing algorithm and solution toolset for delivering valuable insights and prediction models on the connected data generated by various business processes in organizations.

What's Next

As we have seen in the articles published in this series, Apache Spark framework provides the necessary libraries, utilities and tools for unified big data processing application architectures. Whether the data needs to be processed in real time or as a batch, or if the dataset has connections and relationships, Spark makes it easier to work with different types of data. We no longer need to depend on several different frameworks to process and analyze different types of data created and managed in organizations.

If you are looking for a big data solution for applications in your organization, or if you are interested in transitioning to big data and data science areas, Apache Spark is an excellent choice.

References

- Big Data Processing using Apache Spark - Part 1: Introduction
- Big Data Processing using Apache Spark - Part 2: Spark SQL
- Big Data Processing using Apache Spark - Part 3: Spark Streaming
- Big Data Processing using Apache Spark - Part 4: Spark Machine Learning
- Big Data Processing with Apache Spark - Part 5: Spark ML Data Pipelines
- Apache Spark Main Website
- Spark GraphX Main Website
- GraphX Programming Guide
- Spark GraphX in Action, Manning Publications
- Facebook's Comparison of Apache Giraph and Spark GraphX for Graph Data Processing
- Databricks blog article on GraphFrames

About the Author

Srini Penchikala currently works as Senior Software Architect and is based out of Austin, Texas. He has over 22 years of experience in software architecture, design and development. Penchikala is currently authoring a book on Apache Spark. He is also the co-author of Spring Roo in Action book from Manning Publications. He has presented at conferences like JavaOne, SEI Architecture Technology Conference (SATURN), IT Architect Conference (ITARC), No Fluff Just Stuff, NoSQL Now, Enterprise Data World, OWASP AppSec, and Project World Conference. Penchikala also published several articles on software architecture, security & risk management, NoSQL and Big Data topics on websites like InfoQ, The ServerSide, OReilly Network (ONJava), DevX Java, java.net and JavaWorld. He is the Lead Editor for Data Science community at InfoQ.

This is the sixth article of the "Big Data Processing with Apache Spark" series. Please see also: Part 1: Introduction, Part 2: Spark SQL, Part 3: Spark Streaming, Part 4: Spark Machine Learning, Part 5: Spark ML Data Pipelines.