GRAPH DATABASES

Srinath Sridhar & Shweta Anchan

Graph Databases

A graph database or a graph-oriented database is a database that uses graph theory to store, map and query relationships. It basically is a collection of nodes and relationships. Nodes are entities and edges are relationships.

At query time, other databases compute relationships very expensively while graph databases do so rather quickly. They are independent of the total size of the data and hence they handle largely connected graphs and complex queries well. Graph databases are well suited for analyzing interconnections; hence there has been an increased interest in using graph databases to mine data from social media.

Go through the following links to know more about graph database

http://whatis.techtarget.com/definition/graph-database

http://www.slideshare.net/maxdemarzi/introduction-to-graph-databases-12735789

Advantages and Disadvantages

The advantages are:

- Graph databases make it easy to express many kinds of data that requires a lot of fitting in relational databases.
- Searches that are difficult to perform in relational databases are quick and easy in graph databases.
- Graph databases are well suited for irregular, complex data that require mapping in the real world.
- It easily provides for any new kind of data.

The disadvantages are:

- Operations on large amounts of data can be slow.
- It takes up a lot of space.
- As of now, they are not widely used in business environments.
- The data can be described inconsistently; thereby reducing the usefulness of the database.
- Also the data has to be expressed explicitly in relation to other data.
- It is conceptually difficult to understand in the beginning.

Graph databases vs. Relational databases

The primary difference is that in graph databases, relationships are stored at individual record level while in relational databases; the structure is defined at higher level (the table definitions). When it comes to operating on large amounts of data, relational databases are faster. In graph databases, each record has to be examined individually at query time to know the structure of data while it is known ahead of time in relational databases. Relational databases take lesser storage space than graph databases since they do not have to store relationships.

Pregel

Pregel is a framework oriented towards graph-based algorithms. Google developed it. They wanted to create applications that could perform internet-related graph algorithms. Pregel is special in a way that they are made to scale easily on large-scale computer clusters. It is based on Bulk Synchronous Parallel model of programming. Google has not provided any open source implementation of Pregel; hence Apache Giraph and CMU's GraphLab are two open source implementation of Pregel.

How it works?

It follows the Master/Worker architecture. The master initiates message passing between vertices in the graph. The message passing is divided into an iteration of steps called supersteps. At the beginning of each superstep, the worker applies a user-specified compute function on each vertex. The vertex receives the message from the last superstep. The compute function processes the message and sends it to other vertices; which it will receive in the next superstep. There is something called "vote to halt" which deactivates the vertex until it receives the message. Once all vertices decide to "vote to halt", the job terminates.

Pregel is used to solve different problems namely PageRank, shortest path and finding clusters in social networks. It is useful for its ease of programming since its all about vertices and edges and also because of its efficiency on graph problems. It is more efficient that MapReduce when it comes to support iterative computations, since the dataset is in the memory and not written on the disk after each iteration.

The main concepts behind Pregel can be read from the following links

http://www.quora.com/What-are-the-main-concepts-behind-Googles-Pregel

http://www.slideshare.net/shatteredNirvana/pregel-a-system-for-largescale-graph-processing

Titan

Titan is a scalable graph database optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster. Titan is a transactional database that can support thousands of concurrent users executing complex graph traversals in real time. The following section focuses on presenting information required for getting the Titan graph database up and running.

Download Titan:

https://github.com/thinkaurelius/titan/wiki/Downloads

The link mentioned above, contains several versions of Titan. Download the latest or download Titan 0.4.4. The tutorials available online for titan uses Titan 0.4.4. So, you can download that and try out the sample tutorials available online.

Getting started with Titan:

http://s3.thinkaurelius.com/docs/titan/0.9.0-M2/getting-started.html

The link mentioned above, contains a set of commands that you can follow to take an existing graph database and traversing through the available content.

The tutorial uses Gremlin language to interact with Titan. The gremlin that is available in the downloaded Titan package should come with the Tinkerpop integration. If not, download the gremlin exe from the following link

http://tinkerpop.incubator.apache.org/

The commands that you execute by following the tutorial gets you the results in a textual format. Gremlin runs on a terminal and it is difficult to view the results in a graphical format.

You can view the results of your commands in a graphical format by integrating Titan with a third party tool like Keylines. Along with this document, we have included a PDF presenting information on integrating titan with Keylines through Rexster.

GraphX

GraphX is Apache Spark's API for graph and graph-parallel computations. GraphX optimizes the representation of vertex and edge types when they are primitive data types (e.g., int, double, etc…) reducing the in memory footprint by storing them in specialized arrays.

GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph: a directed multi graph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks. GrpahX superseded Bagel.

Example: PageRank

PageRank measures the importance of each vertex in a graph, assuming an edge from *u* to *v* represents an endorsement of *v*'s importance by *u*. For example, if many others follow a Twitter user, the user will be ranked highly.

GraphX also includes an example social network dataset that we can run PageRank on. A set of users is given in graphx/data/users.txt, and a set of relationships between users is given in graphx/data/followers.txt. We compute the PageRank of each user as follows:

```scala
import org.apache.spark.SparkContext
import org.apache.spark.graphx.GraphLoader


object PracticeScala {
  def main(args: Array[String]) {
    val sc = new SparkContext("local[2]", "xyz")


    val graph = GraphLoader.edgeListFile(sc, "/Users/shwetaanchan/Downloads/followers.txt")
    val ranks = graph.pageRank(0.0001).vertices

    val users = sc.textFile("/Users/shwetaanchan/Downloads/users.txt").map { line =>
      val fields = line.split(",")
      (fields(0).toLong, fields(1))
    }
    val ranksByUsername = users.join(ranks).map {
      case (id, (username, rank)) => (username, rank)
    }
    // Print the result
    println(ranksByUsername.collect().mkString("\n"))


  }
}
```

Output:

```
(justinbieber,0.15)
(matei_zaharia,0.7013599933629602)
(ladygaga,1.390049198216498)
(BarackObama,1.4588814096664682)
(jeresig,0.9993442038507723)
(odersky,1.2973176314422592)
```

This shows us that the user justinbieber is highly ranked because he has the most number of followers.

For further information about page ranking using graphx, click on the following link

http://spark.apache.org/docs/latest/graphx-programming-guide.html - pagerank

Neo4J

Neo4J is a graph database storing data in a graph capable of elegantly representing any kind of data in a highly accessible way. It uses a programming language called Cypher.

Cypher is a declarative, SQL-inspired language for describing patterns in graphs. It allows us to describe what we want to do with a graph database without requiring us to describe exactly how to do it. The hyperlink provided in the above paragraph contains a tutorial on how to use cypher in Neo4J.

You can download the tool from the official website and start the Neo4J Community executable application. This starts the Neo4J server and provides a web address that you can click on to interact with the tool in a browser.

Click on the following link to use Neo4j with Spark

http://neo4j.com/blog/getting-started-apache-spark-neo4j-using-docker-compose/