**Topic: Bitcoin Coin-graph**

## TABLE OF CONTENTS

### INTRODUCTION

* 1. Blockchain 1
  2. [Challenges in analyzing parallel processingofgraphs 1](#_TOC_250010)

[LITERATURE SURVEY](#_TOC_250009)

* 1. [IntroductiontoGraphX 2](#_TOC_250008)
  2. [Triangle count implementationusingGraphX 2](#_TOC_250007)
  3. [Triangle count implementationusingGraphFrames 3](#_TOC_250006)

[PROPOSED THEORY](#_TOC_250005)

* 1. [Introduction 4](#_TOC_250004)
  2. Process of transforming raw data intodataframe 4

and perform Triangle count algorithm

[IMPLEMENTATION](#_TOC_250003)

* 1. [Implementation of Trianglecountalgorithm 6](#_TOC_250002)
  2. Implementation of top 30 donorsofWikileaks 6

[RESULTS AND DISCUSSIONS 7](#_TOC_250001)

[REFERENCES 7](#_TOC_250000)

**INTRODUCTION**

* 1. **Bitcoins and Blockchain**

**Bitcoin**is cryptocurrency and worldwidepayment system. It is the first decentralized digital currency, as the system works without a central bank or single administrator. The network is peer-to-peer and transactions take place between users directly through the use of cryptography, without an intermediary. These transactions are verified by network nodes and recorded in a public distributed ledger called a blockchain.[1]

#### Block Chain

The blockchain is a shared public ledger on which the entire Bitcoin network relies. All confirmed transactions are included in the blockchain. This way, Bitcoin wallets can calculate their spendable balance and new transactions can be verified to be spending bitcoins that are actually owned by the spender. The integrity and the chronological order of the blockchain are enforced withcryptography.[2]

### Challenges in analyzing parallel processing of graphs

The following properties of graph problems present significant challenges for efficient parallelism

* **Data-driven computations:**Graph computations are often completely data-driven. The computations performed by a graph algorithm are dictated by the vertex and edge (node and link) structure of the graph on which it is operating rather than being directly expressed in code. As a result, parallelism based on partitioning of computation can be difficult to express because the structure of computations in the algorithm is not known apriori.
* **Unstructured problems:**The data in graph problems are typically unstructured and highly irregular. Similar to the difficulties encountered in parallelizing a graph problem based on its computational structure, the irregular structure of graph data makes it difficult to extract parallelism by partitioning the problemdata.
* **Poor locality:**Because graphs represent the relationships between entities and because these relationships may be irregular and unstructured, the computations and data access patterns tend not to have very much locality. Thus, high performancecanbehardtoobtainforgraphalgorithms,evenonserialmachines.
* **High data access to computation ratio:**Graph algorithms are often based on exploring the structure of a graph in preference to performing large numbers of computations on the graph data. Since these accesses tend to have a low amount of exploitablelocality,runtimecanbedominatedbythewaitformemoryfetches.[3]

# LITERATURE SURVEY

### Introduction to GraphX

**GraphX**is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a newGraphabstraction: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g.,subgraph,join Vertices, andaggregateMessages) as well as an optimized variant of thePregelAPI. In addition, GraphX includes a growing collection of graphalgorithmsandbuildersto simplify graph analytics tasks.

The**Property Graph**is a directed multigraph with user defined objects attached to each vertex and edge. A directed multigraph is a directed graph with potentially multiple parallel edges sharing the same source and destination vertex. The property graph is parameterized over the vertex (VD) and edge (ED) types. These are the types of the objects associated with each vertex and edge respectively. 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. [4]

### Triangle count implementation using GraphX

A vertex is part of a triangle when it has two adjacent vertices with an edge between them. GraphX implements a triangle counting algorithm in the

TraingleCount objectthat determines the number of triangles passing through each vertex, providing a measure of clustering.

TriangleCount object in GraphX counts the triangles passing through each vertex using a straightforward algorithm:

1. Compute the set of neighbors for eachvertex;
2. For each edge compute the intersection of the sets and send the count to both vertices;
3. Compute the sum at each vertex and divide by two since each triangle is counted twice.

Suppose A and B are neighbours. The set of neighbours of A is [B, C, D, E]; the set of neighbours of B is [A, C, E, F, G]. The intersection is [C, E]. The vertices in the intersection are their common neighbours, so [A, B, C] and [A, B, E] are two triangles.[5]

### Triangle count implementation using GraphFrames

GraphFrames is a package for Apache Spark which provides DataFrame-based Graphs. It provides high-level APIs in Scala, Java, and Python. It aims to provide both the functionality of GraphX and extended functionality taking advantage of Spark DataFrames. This extended functionality includes motif finding, DataFrame-based serialization, and highly expressive graph queries.

GraphFrames represent graphs: vertices (e.g., users) and edges (e.g., relationships between users). If you are familiar with GraphX, then GraphFrames will be easy to learn. The key difference is that GraphFrames are based upon Spark DataFrames, rather thanRDDs.

GraphFrames also provide powerful tools for running queries and standard graph algorithms. With GraphFrames, you can easily search for patterns within graphs, find important vertices, and more. Refer to the User Guide for a full list of queries andalgorithms.[6]

**PROPOSED THEORY**

### 3.1Introduction:

In this project we aim to process the information related to bitcoin transactions by extracting substantial records stored as comma separated files and converting them into dataframes, creating tables for the dataframes and joining the tables to ascertain relations between wallet addresses involved in the transactions.

We aim to achieve this by implementing various python libraries such as SparkContext, Sql etc.

### 3.2 Process of converting raw data into dataframe and perform the triangle count algorithm:

**STEP1:**Import as text files vin.csv and vout.csv and create an RDD

**STEP2:**Creating the schema using first line as the header and splitting at comma

**STEP3:**Using the RDD and schema create a dataframe

**STEP4:**Perform corresponding steps 2 and 3 for vin and vout

**STEP5:**Join both the dataframes based on matching the values from vin and vout tables where tx\_hash from vin is equal to hash from dataframe vout and value vout from vin is equal to n from vout dataframe. By doing so, we get the wallet address of all the senders

**STEP6:**Join txid from previously created dataframe with the hash from vout dataframe to obtain the wallet address of receivers

**STEP7:**Now, we have all the wallet addresses in one single dataframe. By applying the select operation, we obtain 2 columns containing only the wallet addresses i.e. publickey. Each element in a row represents a vertex in the graph.

**STEP 8:**If VertexA < VertexB, then make VertexA as sender key. Otherwise make it as a receiver key. Selecting only Source and destination wheresrc<dst.

**STEP 9:**A Dataframe with two columns will be created which are undirected edges

**STEP 10:**Now, using this newly created dataframe containing 2 columns, we create 2 copies of this dataframe, rename the columns and perform join and select operations and get a new dataframe consisting of 3 columns. These 3 columns represent a transaction between 3 addresses which form a triangle

**STEP 11:**Using these 3 columns we find the number of distinct rows, which represents the total number of triangles formed.

## IMPLEMENTATION

The project has been created using Spark1.6.0.

The language used for implementation is python. Determining the language to choose between Scala and Python was a tough choice, since python provides is dynamically typed, less verbose and provides support for multiple machine learning libraries, which were later used to compare the result of the algorithm proposed by us with algorithms and methods already defined in theselibraries.

One such library provided by python is GraphFrame.

The code begins with initializing pyspark and importing the required libraries.

The case study is BitCoins, where we determine a process of how to convert raw data into data frames and also implement the algorithm for Triangle count.

### Implementation of triangle count algorithm:

Taking the .csv files stored at the cluster as arguments to the sc.textFile() we created 2 RDDs, accessing the first 2 records of the RDDs we split the records on occurance of commas.

Using the createDataFrame() we created 2 DataFrames.

Performing the join function on the 2 dataframes and creating one single data frame to get the publicKey(which is the sender wallet address) associated with each transaction. Then performing a join of the new dataframe with vout to obtain the receiver’s wallet address. Adding condition if A<B to set the source and destination wallet (the edges). Creating a new dataframe by the same columns and then renaming the columns to later join the dataframes on column B and then on column A and column C to get thetriangle.

### Evaluate the top 30 donors over the subset period for the Wikileaks bitcoin address

A SQL query is used for retrieving the donors for Wikileaks, where we join the dataframes to get the sender wallet address and then from joining the id of the associated transaction on vout on a specific wallet address of WikiLeaks (1HB5XMLmzFVj8ALj6mfBsbifRoD4miY36v), we get the transaction of all the donors of WikiLeaks along with the amount which was then summed up, and then displaying them in descending order. Finally, the top 30 donors are displayed using the head() function.

## RESULTS AND DISCUSSIONS

We have taken a sample of starting 300,000 records of vin.csv and vout.csv data from /data/bitcoin folder and execute the code on the sample. The count of triangles we got from executing Finding\_Triangles.py came out to be**25238.**To verify our result we executed GraphFrameImpl.py to find the count of triangles using the graphframe library which results the same,**25238**which verifies our result. The performance indication was to verify the answer of our algorithm with the GraphFrame function of triangle count providing the edges andvertices.

For the additional project goals, we have compared the performance of Triangle count technique with GraphFrame implementation as GraphX was not supported by Spark with Python. Apart from that we have also evaluated the top 30 donors for the Wikileaks bitcoinaddress.

## REFERENCES

1. https://en.wikipedia.org/wiki/Bitcoin

#### https://bitcoin.org/en/how-it-works

1. Andrew Lumsdiane and Douglas Gregor, Bruce Hendrickson and Jonathan Berry,*Challenges In Parallel Processing*, Indiana University, Indiana, December 2006
2. https://spark.apache.org/docs/latest/graphx-programming-guide.html
3. <http://note.yuhc.me/2015/03/graphx-triangle-count-label-propagation/>
4. https://graphframes.github.io/