SparkGalaxy: Workflow-based Big Data Processing

Sara Riazi
Department of Computer and Information Science
University of Oregon

Abstract

We introduce SparkGalaxy, a big data processing toolkit that is able to encode complex data science experiments as a set of high-level workflows. SparkGalaxy combines the Spark big data processing platform and the Galaxy workflow management system to offer a set of tools for graph processing and machine learning using a novel interaction model for creating and using complex workflows. SparkGalaxy contributes an easy-to-use interface and scalable algorithms for data science. We demonstrate SparkGalaxy use in large social network analysis and other case studies.

1 Introduction

Big data processing becomes more and more important as significant increases in storage capacity makes it possible for systems to keep track of their users and events, which results in large amounts of data stored as heterogeneous data sources. Understanding this data is important for scientists and business owners as it reveals the structure and dynamics of the underlying system. However, in addition to having knowledge about data sciences, mining big data requires expertise in the area of high-performance computation and parallel processing, which is not trivial for a considerable group of data scientists.

Although recent advances in data-parallel computation give the users a high-level abstraction through procedural and relational APIs, dealing with programming environments and encoding whole experiments as a set of programs and scripts can decelerate research progress.

We observe that a successful practice to address this problem is to provide a web-based and easy to use toolkit which encapsulates details, and enables its users to encode complex research experiments as high-level diagrams. One of the very successful examples is Galaxy. Galaxy[1] is an open-source web-based platform for biomedical projects. We describe Galaxy in more details here in Section 2.4.

In this project, we introduce SparkGalaxy, a workflow-based toolkit built on top of Galaxy that provides a set of components for analyzing and mining big data. Using the

[1] https://usegalaxy.org/

toolkit, data scientists can construct complex workflows for their experiments, and run their experiments on cluster systems.

The remainder of this report is organized as follows. In Section 2, we describe the background. In Section 3, we introduce our SparkGalaxy toolkit and its components, and in Section 4, we provide some case studies to verify the potential of SparkGalaxy for describing complex experiments. Finally, in Section 5, we describe future work and conclude this report.

2 Background

2.1 Data-parallel distributed processing

One of the most significant advances in distributed data processing is the Map-Reduce programming model [5]. In Map-Reduce, data is converted to key-value pairs and then partitioned to nodes. A Map-Reduce system consists of a set of workers that are coordinated by a master process. The master process assigns partitions to workers, and then workers apply a user-defined map function to the key-value pairs, resulting in intermediate key-value pairs stored on the local disks of workers. The intermediate key-value pairs are passed to another set of workers that group the key-values by keys and apply a user-defined reduce function on the group of values associated to a particular key. The workers then apply the reduce function, and store the output on their local disks, so one can combine different partitions of the output together and create a single output file, or pass the output as the input to another map-reduce call.

For using the map-reduce programming model, the problem has to be defined as a set of map and reduce functions, so we cannot alternate the orders, for example, by applying two map functions and then a reduce function. Although this problem can be solved by constructing a more complex map function that consists of more than one map function, it reduces the reusablity and modularity of the model.

Another restriction of Map-Reduce model is that we can only apply the Map-Reduce model on homogeneous data sources, which limits the usage of Map-Reduce models when we have to join the information from different sources. This problem has been addressed by variant models of Map-Reduce, such as Map-Reduce-Merge [23].

2.2 Spark big data processing framework

Apache Spark[2] is a fast growing framework for large data processing. Spark provides interfaces for popular programming languages such as Python, Scala, and Java. Moreover, it includes many libraries for machine learning, graph algorithms, streaming processing, and relational data processing. Spark supports a distributed architecture, in which an application is running as set of processes. The main program, called the driver, consists

[2]https://spark.apache.org/

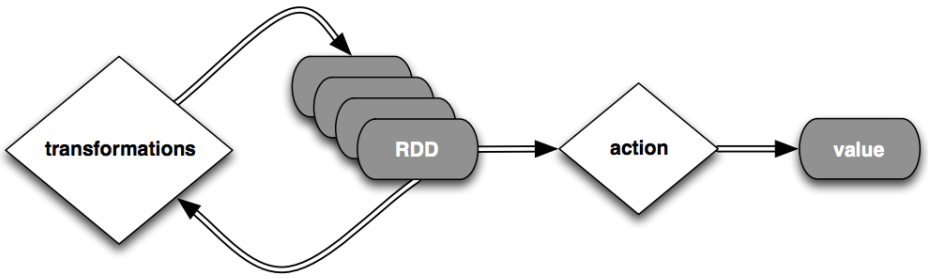

Figure 1: Schematic diagram of the relation of transformations and actions on RDDs.

of an object called *SparkContext* which coordinates the execution of the application's processes on Spark worker nodes through a Spark master, which manages the workers. The most important concept in Spark is its resilient distributed datasets (RDDs). RDDs [24] are immutable collection of objects that are partitioned across different Spark nodes in the network. There are two main instructions to create RDDs: through loading a distributed file from HDFS, for which Spark maps each block of HDFS to a partition of RDDs, or by parallelizing an object across different Spark nodes.

An RDD is transformed to another RDD using transformation instructions such as *map* and *filter*. Spark introduces the notion of *lineage* for RDDs, which means that Spark keeps the information about how it derives a new RDD through the transformation of the other RDD. In this way, if one partition gets lost, Spark has enough information to rebuild it. Lineage makes Spark fault-tolerant and efficient because it does not have to keep redundant blocks of data to provide fault-tolerance. Transformations in Spark are lazy, which means that Spark does not apply transformations immediately. It, instead, constructs a directed acyclic graph (DAG) of data parts and transformations followed by final steps as actions. Then it executes the formed DAG by sending it as several tasks to Spark nodes. The actions in Spark reduce RDDs to values. For example, *count* computes the number of records in RDDs, so it needs all the transformation to be applied first, and then it returns the result. Figure 1, shows how the transformations and actions applies to RDDs.

The other benefits of lazy transformation is that Spark can optimize the operations. For example, suppose we have an RDD of strings and we want to filter the RDD based on the appearance of word "Oregon" and then we have another filter of the intermediate RDD, which looks for word "Eugene", and finally, we want to count the number of objects in the final RDD. When we trigger the transformation by calling the count, Spark jointly applies both filters and count the number of occurrence without constructing the intermediate RDDs, this optimization will speed up the evaluation of consecutive RDD operations.

Another data abstraction in Spark is the dataframe [1]. A dataframe is a distributed collection of data organized into named columns, which is conceptually equivalent to a table in a relational database. Dataframes can be manipulated using relational APIs,

which offers richer optimizations. Dataframes can also be viewed as RDDs of *Row* objects, which enables users to process dataframes using procedural APIs offered by RDDs. Similar to RDDs, the relational operation are lazily applied on dataframes.

Dataframes supports a domain-specific language for relational operations, including *select*, *filter*, *join*, and *groupBy*. Dataframes also can be registered as relational tables, which enables users to perform pure SQL queries on them.

2.3 Graph parallel processing

Many real-world problems are described using networks and graphs such as social networks, Internet maps, and protein interactions. These graphs may scale to billions of nodes and edges. The complexity of graph algorithms is usually polynomial in the number of vertices of the graph. As a result, running such algorithms over very large graphs is very time consuming. Graph parallel processing frameworks have been introduced to process these very large graphs. Pregel [15], GraphLab [14] and Spark GraphX [22] are all examples of graph parallel processing frameworks that have been widely used recently.

Pregel [15] is a bulk synchronous message passing abstraction such that it iteratively runs the program associated with each vertex simultaneously. In each iteration, it gathers all messages from the previous iteration, and prepares messages for the next iteration. The program terminates when there are no more messages and every vertex votes to halt. Apache Giraph[3] is an open source implementation of Pregel, and works over the Hadoop MapReduce paradigm.
GraphLab [14] is a commercial machine learning and graph processing framework based on the Gather-Apply-Scatter (GAS) model, similar to Pregel. In the GAS model, the algorithm runs over three stages: data preparation, iteration, and output. In the data preparation stage, it initializes the values of vertices and edges, and then executes the iteration stage until all vertexes vote to halt, then it gathers the output. Each iteration itself consists of gather, apply, and scatter. A vertex gathers the values of its adjacent vertexes and edges and then applies its partial algorithm and scatters the results. A very common example of an algorithm written GAS paradigm is PageRank.

GraphX is another large-scale graph processing framework developed on top of Apache Spark. Since Spark is a data-parallel computation system, GraphX implements graph operations based on data-parallel operations available in Spark. GraphX represents graphs using an RDD for vertices and another for edges [22]. However, handling graphs in a data-parallel computation system is more complex than map-reduce operations since the vertices should be processed in the context of their neighbours. To address that, GraphX introduces the triplet concept, which joins the structure of vertices and edges. Each triplet carries the attributes of an edge and the attribute of vertices that incident with that edge. Therefore, by grouping triplets on id of the head or tail vertices, one can access the attributes of all the neighbors of each vertex. Moreover, since the triplets are distributed, if the neighbors of a vertex are located on different machines, then we

[3]http://giraph.apache.org

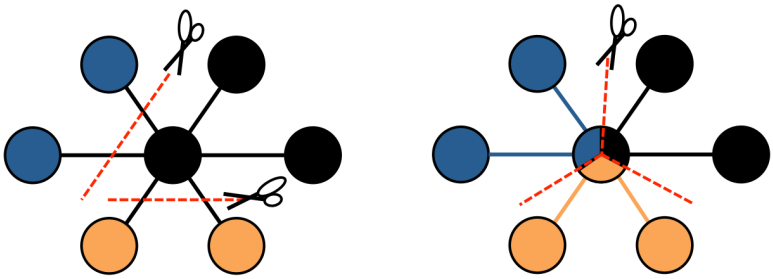

Figure 2: Edge-cut vs. Vertex-cut. The colors depict different machines. The left figure shows a edge-cut, in which four edges are assigned to two different machines. The right figure, shows a vertex-cut, in which a vertex is assigned to three different machines [22].

have a communication overhead to construct the groupBy result. Therefore, strategies for distributing graphs over different partitions become important in terms of communication overhead and storage overhead. Two main partitioning strategies for graphs exist: edge-cut and vertex-cut [22]. In edge-cut, the vertices of a graph are evenly assigned to different machines, so the edges may span across different machines. We can optimize the partitioning to reduce the number of edges that span on different machines (reduce the number of cuts). Vertex-cut, on the other hand, evenly distributes the edges over the machines and may keep multiple copies of a vertex on different machines if the edges that incident with the vertex are assigned to different machines. Here, the communication overhead is to synchronize the information of a copied vertex on the machines that store the copies. Experimentally it has been shown that real-world graphs have better vertex-cut than edge-cut [8, 11].

As mentioned earlier Gather-Apply-Scatter (GAS) model is a widely used paradigm for graph parallel processing. GraphX implements this paradigm on top of Spark data-parallel computation. GraphX implements gathering attributes from neighbours using the groupBy operation on triplets, and then applies map-reduce functions on the grouped data associated with each vertex. This operation results in another vertex RDD, which includes the aggregated message in each vertex. By joining this RDD to edge RDD, GraphX constructs a new triplet collection that can be used in the next iteration. GraphX offers GAS paradigm through *aggregateMessage* method of Graph class.

2.4 Galaxy

Galaxy [7] is a public and user-friendly data integration and workflow management system, mostly used in biomedical sciences. Recently, it also has been used for social sciences [16], but the provided tools are based on common statistical algorithms, and are not intended for big data processing. Galaxy does not have any abstraction for graph data. Galaxy enables users to graphically describe workflows by drawing pipes to connect tools. Then, the system manages the execution of the workflow by running the tools over input datasets in a defined order. Galaxy is supported by an app-store called Toolshed, which

```
<?xml version="1.0"?>
<tool name="Head" id="head">
    <description>
        Print top n lines of data
    </description>
    <command interpreter="python">head.py $input $output $num </command>
    <inputs>
        <param format="txt" name="input" type="data" label="Input"/>
        <param size="10" type="text" value="10" name="num" label="Top lines:"/>
    </inputs>
    <outputs>
        <data format="input" name="output" label="Output" />
    </outputs>
</tool>
```

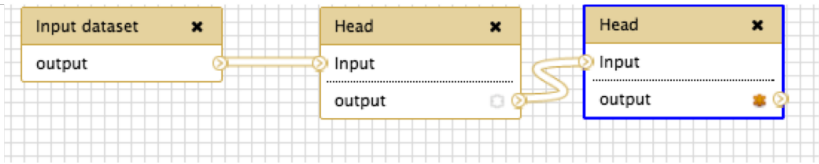

Figure 3: Example of XML files that describe Galaxy tools and a simple workflow produced using Galaxy.

supplies different, mostly bioinformatic tools, for gathering and processing data. The toolshed allows users to introduce new tools and include them in the workflows. Once a workflow has been created, Galaxy stores it, so it can reproduce the experiment. Galaxy runs each tool as a separate program, and provides it with the output of the previous tool in the workflow as the input of the program, and takes its output for the next step in the workflow.

Galaxy tools are described using XML files. An XML file specifies the input, parameters and output of the tools and also specifies how the tool should be executed. Figure 3 shows an example XML file and describes a simple tool that outputs top lines of a ASCII file. It also shows a simple workflow using the described tool.

To run a tool, Galaxy creates a working directory for the tool and prepare the input and output addresses and run the tool. It gives the user an option of keeping the output in a permanent data history. The data in the history are deleted on demand.

Many bioinformatic applications are data-intensive, so running the Galaxy workflows in the cloud increases the resources available to each tool and speeds up the evaluation of workflows. In such cloud-based environment, tools in a workflow that do not depend on each others run in parallel on different machines [13]. In these platforms Galaxy is offered as service on purchasable computational resources. BioBlend [20] is another approach for parallelizing Galaxy workflows, and offers a rich API for accessing Galaxy workflows and jobs and run them over clouds. Nevertheless, these platform and frameworks do not parallelize the execution of each tool over different machines, so many data science experiments that are usually expressed as a pipeline of tools and scripts would not benefit from these cloud-based scalable approaches.

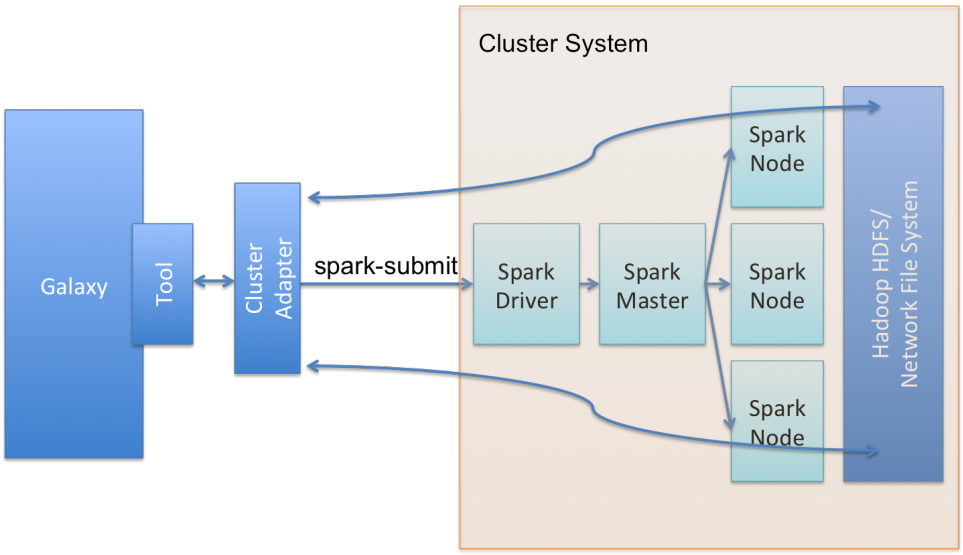

Figure 4: The architecture of SparkGalaxy. The Spark-based tools in Galaxy interact with Spark nodes on the cluster system using a cluster-adapter.

3 SparkGalaxy Toolkit

In this section, we introduce SparkGalaxy, a workflow-based big data processing toolkit. The SparkGalaxy toolkit is a set of Galaxy compatible tools that offer the rich procedural API in a higher level abstraction. Using SparkGalaxy, we can construct complex data science experiments as a workflow of Spark-based tools.

Figure 4 shows the general architecture of SparkGalaxy. Each Galaxy tool submits a Spark application to cluster systems through cluster-adapter or runs it on the local machine. The cluster-adapter provides tools by the address of Spark master node, which can be run on the cluster or on the local machines.

The input data provided by Galaxy must be accessible to Spark applications and output data generated by Spark applications must be accessible to Galaxy. The cluster-adapter also responsible for this data accessibility, so we may need different cluster-adapters depending on the underlying cluster system.

In addition, Galaxy expecting the input data to be stored as single locally stored file in a conventional file-systems (not distributed file system such as HDFS), which is inconsistent with the fact that Spark partitions data into multiple files, which may also be distributed over several machines.

To address this inconsistency, Pireddu et al. [17] introduce a functional and extensible integration layer, which enables the users of Galaxy to combine Hadoop-based tools with conventional tools in their workflows.

Their adaptation layer combines the HDFS address of input data files as a *pathset* and pass the constructed pathset to a Hadoop-based tool, which outputs another pathset as results. This resulting pathset can be input of another Hadoop-based tool. We build on this indirect referencing and introduce *Metafile* as the input and output format of

Spark tools. A Metafile is an XML description of the objects, the address type, and object address. Using the information about the address type, and the cluster-adaptor can determine whether the object is stored locally, on HDFS, or on a network file system, so it posts the application to the requested cluster-system or local machines if the data is available to it. Moreover, to avoid data migration, the address type is used for allocating space for the output data at the same file server as the input data.

Metafiles also include the schema of the data, which helps users attain general understanding about the underlying data because only Metafiles are accessible to users through the Galaxy experiment history.

3.1 Interaction Model

The ultimate goal of SparkGalaxy is to provide a workflow-based environment that is capable of encoding complex data science experiments. Each tool is a Spark application that manipulate distributed collection of objects in format of RDDs or dataframes, so we can consider that each Spark-based tool is a complex map, reduce, or join function that transforms RDDs or dataframes to another RDD or dataframe, or reduces them to a scalar value.

Loading and storing typed collection objects such as RDDs reduces the generalization of Spark-based tools because RDDs have to be manipulated differently based on the type of the object they encapsulate. For example, an RDD of a String array is different from an RDD of a Double array. This becomes more complicated when we have more structured objects such as tuples.

Therefore, to boost the generalization, each Spark-based tool expects the input to be in a named column format such as CSV files. The Spark application loads the input CSV file into a dataframe, and applies a set of functions to it and transforms it into another dataframe. Finally the application stores it as another CSV file. It is worth mentioning that these CSV files are multi-part files, and tools actually expect a Metafile as input that contains the schema of these CSV files and their addresses, and produces another Metafile as output. The schema of an output Metafile may be different from the schema of the input Metafile. We use the Spark-CSV library[4] for reading/writing dataframes from/to CSV files.

Tools may expect more than one dataframe as the input and may produce more than one dataframe as the output. For example, the tools that work on graphs expect a graph as input, and graphs are representable as two dataframes, one for vertices and another for edges.

3.2 SparkGalaxy components

The SparkGalaxy tools are grouped into cluster management tools, input/output tools, graph tools, and machine learning tools. All SparkGalaxy tools return a log file in addition to their expected output. This log output is a single text file understandable by Galaxy.

[4]https://github.com/databricks/spark-csv

The log files usually includes a small sample of output dataframes and execution log of the tool. For simplicity, we do not explicitly mention the log output in the description of the tools. Next, we describe SparkGalaxy components in more detail.

3.2.1 Cluster management tools

This set of tools helps the user to manage the execution of Spark master and workers on the cluster system.

3.2.2 Input/output tools

Input/output tools are used to convert single-file data into dataframes and also to convert them back to single-file data.
PathToMeta: We expect SparkGalaxy's users to upload their big data files directly to the clusters instead of uploading through Galaxy's Web interface. Since the input and ouput of SparkGalaxy tools has a Metafile format, the PathToMeta tool can be used as the initial tool of any workflow to generate a Metafile from an already stored big data.

Data2RDD: The Data2RDD tool converts single-file data to a dataframe. It can be used for loading CSV files into general dataframes or special dataframes used for machine learning tools, which expects the input dataframes to have *label* and *feature* columns. Data2RDD tool is also able to load *libsvm* data format into machine learning-specific dataframes.

RDD2Data: The RDD2Data tool converts a dataframe into single-file data readable by Galaxy.

Data2Graph: The Data2Graph tool takes a Metafile pointing to an edge-view file of a graph, loads it into a Spark's graph structure, and returns another Metafile that points two dataframes: one dataframe for vertices of the graph, and another dataframe for the edges of the graph. The output graph is used by graph tools.

3.2.3 Datasets

These tools output predefined Metafiles for CSV files and graph datasets available on the cluster system.

3.2.4 Graph tools

PageRank: PageRank is a well-known graph vertex ranking algorithm introduced by Google for ranking Web pages. SparkGalaxy's PageRank takes a graph represented as two dataframes of its vertices and edges, and produces a dataframe with two columns of vertex ID and rank value, in which the rank values is computed using the PageRank

algorithm.

DegreeCount: Similar to PageRank tools, DegreeCount takes a graph and returns a dataframe of vertex ids and degree count values.

LargestCC: LargestCC takes a graph and outputs the subgraph of its largest connected components, so the output of LargestCC is another graph represented using two dataframes as described earlier.

Subgraph: Subgraph is a functionality of the GraphX's graph abstraction that constructs a subgraph of the original graph. The subgraph function lets the user provide either an edge or a vertex indicator function. The purpose of the indicator function is to determine whether the given edge or vertex belongs to the resulting subgraph. The subgraph tool is based on this subgraph function, and takes a dataframe that consists of two columns of vertices and boolean values, then it returns two subgraphs that are complement of each others. The first subgraph consists of the vertices that their corresponding value is true, and similarly, the second subgraph consists of the vertices that their values are false.

GraphClustering: The graph clustering tool takes a graph as its input and returns two outputs. The first output is a graph called cluster graph. Cluster graph is a graph such that the attribute of each vertex is the cluster number of that vertex. The other output of the graph is dataframe of vertex IDs and cluster numbers. GraphClustering includes Spectral, PCA, and PIC algorithms for clustering as well as a random clustering algorithm which uniformly assigns the vertices to the clusters. We describe the clustering algorithms we developed in more details in Section 3.3.

ClusterEval: ClusterEval includes two clustering metrics: modularity [4] and normalized cut [19]. This tool takes a cluster graph (as described in GraphClustering) and computes the modularity and normalized cut. The ClusterEval act similar to a reduce function that takes two dataframes and returns some values.

TriangleCount: Triangle count is an important measure in graph and network analysis. A triangle count of a vertex is the number of all complete subgraph with three nodes that the vertex is a member of the subgraphs. The TriangleCount tool takes a graph and returns a dataframe containing the traingle count for each vertex, so the columns of the output dataframe are the vertex id and triangle count.

3.2.5 Relational tools

Query: Query tool runs an SQL query over the input dataframes. In order to run a query over a dataframe, it first register the input dataframe as a relational table with the given

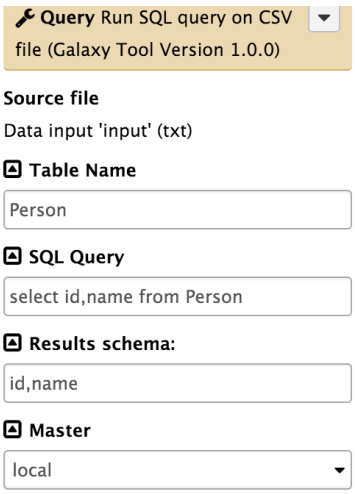

Figure 5: The Query tool expects a table name and query on the given table name. Providing the Query tool with the output schema is optional.

name in the parameters, and then executes the query on the relational table. Figure 5 shows the parameter of page of the Query tool and an example SQL query. Query tool also expects the schema of the output dataframe in order to construct appropriate named columns. The given names are specifically useful when we want to run other queries on the output dataframe.

JoinQuery: JoinQuery tool is similar to Query except it accepts two dataframes as inputs, so we can run join queries on both dataframes. Similar to Query, we have to provide names for the tables and a schema for the results, and the SQL query. JoinQuery is specifically useful when we want to combine the information of two dataframes.

3.2.6 Machine learning tools

These set of tools provides the user with machine learning tools available on Spark. Here we only describe SVMLearner and SVMClassifer, the other classification tools have the same format.

SVMLearner:This tool takes a dataframe as training data, which includes a *label* column and *feature* column, and learns a SVM model from the training data. It ouputs the learned model. The tool's log includes the classification accuracy on the training data.

SVMClassifier: SVMClassifier takes a learned model from SVMLearner and a dataframe containing the test data, and outputs another dataframe that includes true labels as well as predicated ones. The tool's log file includes the accuracy of the learned model on test data.

3.3 Graph clustering

In this section, we describe our implementation of graph clustering algorithms using Spark.

3.3.1 Spectral clustering

Graph **G** is an undirected graph that is represented using its adjacency matrix **A** The Laplacian of **G** is defined using following relation:

$$\mathbf{L} = \mathbf{D} - \mathbf{A} \tag{1}$$

where **D** is a diagonal matrix, in which D_{ii} is the degree of node i in the graph. **L** is symmetric, positive semi-definite matrix. The eigensystem of **L** is all pairs of $(\mu_i, \mathbf{v}_i)$ for which the following equation is true:

$$\mathbf{L}\mathbf{v}_i = \mu_i \mathbf{D}\mathbf{v}_i \tag{2}$$

This relation is a generalized eigensystem. The eigenvector corresponding to the second smallest eigenvalue of a graph's Laplacian matrix can be used to partition the graph, while the eigenvector corresponding to the largest eigenvalue of the graph only shows the connectivity. We can also construct the Markov transition matrix of graph **G** as : $P = \mathbf{D}^{-1}\mathbf{A}$. Suppose u_i and λ_i are the eigenvectors and eigenvalues of P, then we can show that

$$\lambda_i = 1 - \mu_i \quad and \quad \mathbf{u}_i = \mathbf{D} * \mathbf{v}_i \tag{3}$$

Because the dominant eigenvectors of **P** are related to the smallest eigenvectors of **L**, spectral algorithms usually use matrix **P** for graph partitioning because finding dominant eigenvectors is more efficient than finding smallest eigenvectors.

Spectral clustering finds two partitions in a graph by partitioning the nodes in graph based on the entry values of their second dominant eigenvector of transition matrix **P**. They partition the nodes into parts using different cut methods such as ratio cut, bispectral cut, gap cut, or normalized cut [21, 19]. A cut is a set of edges that are removed if we disjoint the two partition of the graph. The ratio cut is the most accurate cut method, in which for every cut, the goodness of cut is evaluated based on the cut value, and then, it selects the best cut. This cut has high computational complexity and practically has no usage, especially for large graphs. The bispecteral cut, on the other hand, is a fast method in which the median of eigenvector entries is selected as a pivot, and then, all the nodes corresponding to entries less than the pivot become the member of one partition and the other partition includes all the remaining nodes.

The main problem of bispectral cut is that it partitions the graph into balanced equal-size parts, which may not show the clusters of the graph.

Gap cut sorts the entries of the second dominant eigenvector, and then finds the entry corresponding to maximum gap between two consecutive sorted entries. Gap cut selects this entry as a pivot, and similar to bispectral cut, partitions the graph into two parts.

Gap cut may find significantly unbalanced partitions, which is not useful in practice. Therefore, Shi and Malik [19] introduce normalized cut, which describes how partitions are relatively balanced in term of cut and total connections from each partition to all vertices of a graph:

$$Ncut = \sum_k \frac{Cut(A_k, \mathcal{V} - A_k)}{Assoc(A_k, \mathcal{V})}, \tag{4}$$

where $\mathcal{V}$ is the set of all vertices of the graph, and A_k is the set of nodes in partition k. *Cut* is the total number of the edges from A_k to $\mathcal{V} - A_k$, and *Assoc* is the total number of the edges from A_k to $\mathcal{V}$.

Shi and Malik [19] also show that finding the Gap cut on the second dominant eigenvector of transition matrix minimizes normalized cut.

The Power method is a well-known method for finding the dominant eigenvector of any square matrix as well as transition matrix $\mathbf{P}$ by recursively multiplying the matrix by a vector:

$$\mathbf{v}^{t+1} = \mathbf{P} * \mathbf{v}^t, \tag{5}$$

where $\mathbf{v}^0$ is a normalized random vector. It is easy to show that vector $\mathbf{v}$ in Relation 5 converges to the dominant eigenvector of matrix $\mathbf{P}$.

Now, suppose that we know that dominant eigenvector of matrix $\mathbf{P}$ is vector $\mathbf{u}$ then we can show that if we update $\mathbf{v}$ in each iteration using Gram-Schmidt algorithm:

$$\mathbf{v}^t = \mathbf{v}^t - (\mathbf{u}^T.\mathbf{v}^t)\mathbf{u} \tag{6}$$

Then the recursive relation converges to the second dominant eigenvector of the matrix. Gram-Schmidt orthogonalizes vector $\mathbf{v}$ to vector $\mathbf{u}$ by subtracting the projection of $\mathbf{v}$ on $\mathbf{u}$ from $\mathbf{v}$.

Therefore, we use a combination of the Power method and Gram-Schmidt to efficiently find the second dominant eigenvector of Markov transition matrix $\mathbf{P}$.

For implementing the spectral clustering using Spark API, we need matrix-vector multiplication, which is the main computation of Relation 5 for computing the eigenvectors and spectral clustering. Here we describe how we can use Spark Gatter-Apply-Scatter (GAS) model to efficiently compute matrix-vector multiplication $\mathbf{w} = \mathbf{P}^T.\mathbf{v}$. Suppose matrix $\mathbf{P}$ has non-zero elements P_{ij} if and only if there exist an edge between node i and node j, and suppose that we assign value v_k to node k of the graph, and we want to compute $w_i = \sum_j P_{ij}v_j$. Since P_{ij} is only non-zero if node i and node j are neighbours, we can compute the summation as $w_i = \sum_{j \in N(i)} P_{ij}v_j$, where $N(i)$ is the set of neighbours of node i. The summation result w_i is stored at node i in the graph. Therefore, to compute w_i in the GAS model, every node sends their value v_i multiplied by the edge weight P_{ij} to their neighbors and the destination node sum up all received messages and construct w_i. Then we can collect all the values as the result of the multiplication. Spark supports this message passing using *aggeragteMessage*:

```
def gram_schmidt_simple(g: Graph[(Double,Double), Double]): Graph[(Double,Double), Double ] = {
  val v1_dot_v2 = g.vertices.map{ case (id,(f1,f2))=> f1*f2}.reduce((f1,f2)=>f1 + f2)
  val v1_norm = g.vertices.map{ case (id,(f1,f2))=> f1*f1}.reduce((f1,f2)=>f1 + f2)
  val coef = v1_dot_v2 / v1_norm
  val orthogonal_g = g.mapVertices{ case (id, (v1, v2)) => (v1, pow(v2 - (v1 * coef ), 2) )}
  val v2_norm = orthogonal_g.vertices.reduce{ case ( (id1,(f1,f2)), (id2, (v1,v2)) )
                    =>  if (id1> id2) (id1,(f1,f2 + v2)) else (id2, (f1,f2 + v2))}._2._2
  val orthonormal_g = orthogonal_g.mapVertices{ case (id, (f1,f2)) => (f1/v1_norm, sqrt(f2)/v2_norm)}
  orthonormal_g
}
```

Figure 6: Gram-Schmidt implementation using Spark GraphX.

```
graph.aggregateMessages[Double](
        sendMsg = ctx => ctx.sendToSrc(ctx.attr * ctx.dstAttr),
        mergeMsg = _ + _,
        TripletFields.Dst),
```

After computing the dominant eigenvectors, we use the kmeans algorithm to partition them into two groups, which constructs two partitions of the graph, also used by PIC clustering [12]. Figure 6 shows an implementation of Gram-Schmidt algorithm using Spark GraphX. We can combine the Map-Reduces for computing *v1-dot-v2* and *v1-norm* to decrease the number of passes over data, but, here, we compute them as separate Map-Reduces for simplicity.

3.3.2 PIC clustering

Power iteration clustering (PIC) [12] is a graph clustering algorithm. It uses power method, Relation 5 to find the dominant eigenvector of transition matrix. However, it stops early before convergence. Lin and Cohen [12] show that this unconverged vector carries information about graph clusters. To find the partitions, PIC uses the kmeans algorithm to split the unconverged vectors into groups of vertices.

The Spark machine learning library provides PIC clustering, which we used as the graph clustering algorithm. The Spark implementation of PIC uses similar matrix-vector multiplication for applying the power method algorithm.

3.3.3 PCA clustering

The principal components (PCs) of transition matrix $\mathbf{P}$ can be used for graph clustering [3]. For PCA clustering, we compute the PCs using the provided Spark API, and then applies the kmean algorithm on the PCs to find the graph partitions. The main limitation of this implementation is that Spark cannot efficiently represent a big square matrix, so this approach is not applicable to the graphs with more 65,535 nodes.

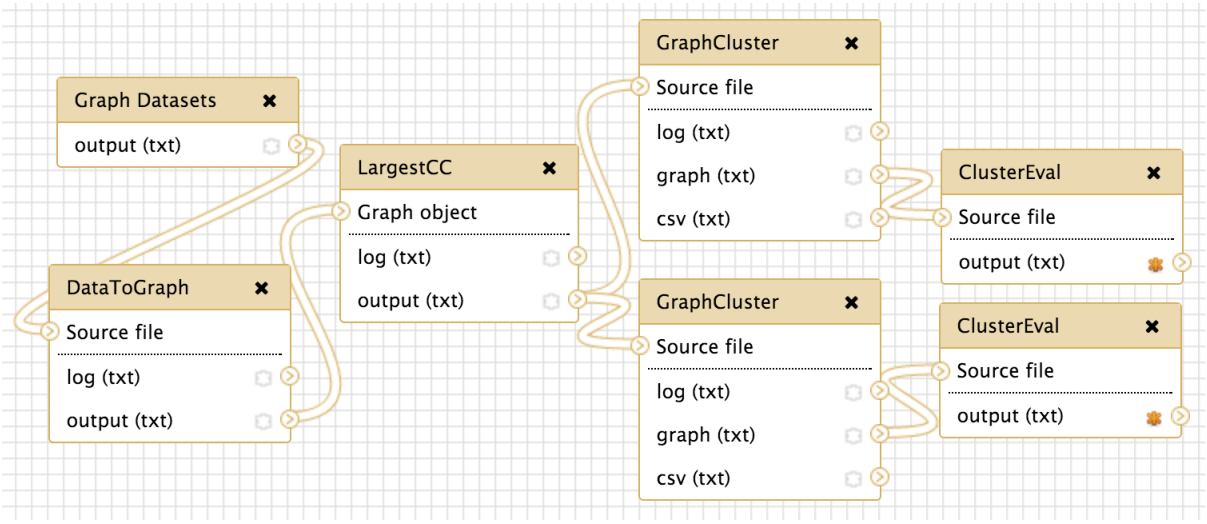

Figure 7: Comparing the performance of different clustering algorithms to reveal the high-level structure of the input graph.

4 Case studies

In this section we provide some case studies to show the capability of SparkGalaxy to represent and run complex data science experiments.

4.1 Graph clustering

Graph clustering is important in many areas specially in social network analysis [6], in which it used for community detection and studying the structure of underlying social network. Figure 7 shows a workflow that loads graph data sets from their edge-view files, finds the largest connected components of the graph, and then applies two different clustering algorithms on the largest connected components of the graphs and evaluates the clustering results using modularity and normalized cut (Eq. 4) metrics.

Modularity compares the existing structure with the expected structure that may exist randomly. Modularity of a set of clusters $\mathcal{C}$ is defined as [4]:

$$M(\mathcal{C}) = \sum_{C \in \mathcal{C}} \Big[\frac{E(C)}{m} - (\frac{\sum_{v \in C} d_v}{2m})^2 \Big], \tag{7}$$

where m is the total number of edges in the graph, d_v is the degree of vertex v, and $E(C)$ is the number of edges inside cluster C. Figure 8 shows the Map-Reduce implementation of modularity using Spark GraphX.

4.2 Graph mining

Mining graph metrics such as degree distribution, triangle count or vertex rank is very common in social network analysis [18, 9]. Here, we show that how we can capture the

```
//In degClusterGraph, the attribute of each vertex is
// a tuple of the vertex degree and its cluster id
val existing = degClusterGraph.triplets.map(t=>
  if (t.srcAttr._2 == t.dstAttr._2) {0.5} else {0.0}).reduce(_+_) / m
var expected = 0.0
for (i<-0 to numCluster - 1) {
  val part = degClusterGraph.vertices.map{case (v,(d, c))=>
    if (c == i) {d} else 0.0}.reduce(_+_) / (2*m)
  expected = expected + pow(part,2)
}
val modularity = existing - expected
```

Figure 8: Modularity implementation using Spark GraphX.

joint information about vertex rank metric using PageRank algorithm and triangle count on a graph similar to experiments done in [2]. The workflow in Figure 9 loads a graph from a edge-view file and applies PageRank tool and TriangleCount tool separately on the input graph. To combine the information from these two tools, we used the JoinQuery tool. JoinQuery register the output of PageRank and TriangleCount as relational tables *ranks* and *counts*, respectively and run the following SQL query to find the traingle count of the top 100 vertices with higher ranks:

```
SELECT ranks.vertex,ranks.rank, counts.count
FROM ranks,counts
ORDER BY ranks.rank LIMIT 100
```

The output of the JoinQuery tool is not big data, so we can use RDD2Data to combine the multi-part data into a single file, and let the Galaxy to store in its experiment history.

4.3 Machine learning

Many data science problems use machine learning methods such as classification, clustering, or regression to produce an accurate model to describe data. Figure 10 depicts a workflow that loads training and test data and learns SVM models using training data and evaluate the learned model on the test data.

The learning tools are able to select hyper-parameters through cross-validation or train/validation split. To do so, learning tools expect the user to provide all the possible values for each hyper-parameter as a comma separated list. The learning tool creates a parameter grid from the provided values and fit the best possible model.

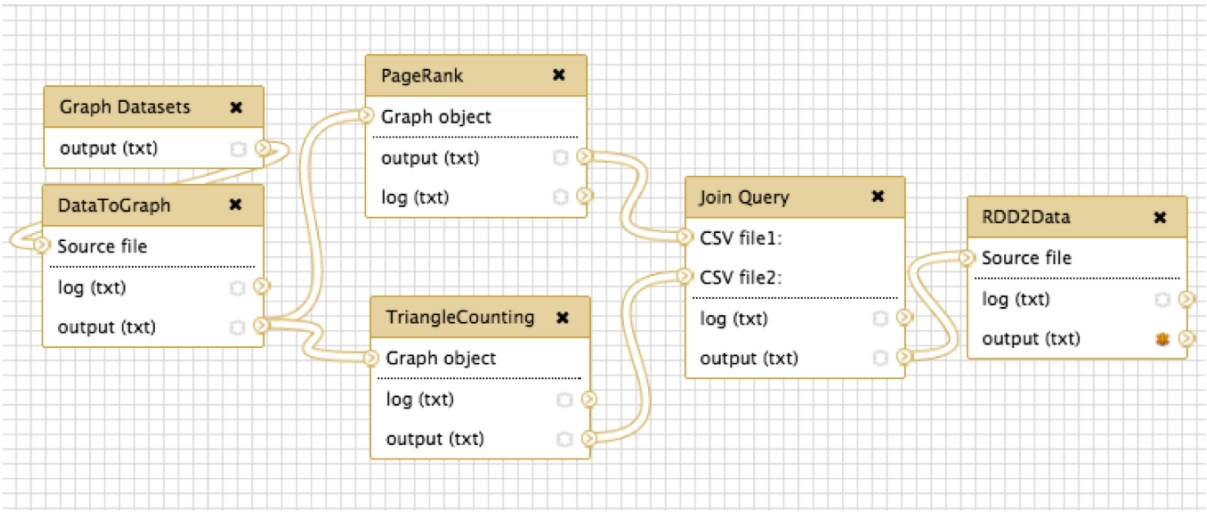

Figure 9: Computing graph metrics using PageRank and TriangleCount and combining the output of them using JoinQuery to have joint inference about rank and triangle count of vertices.

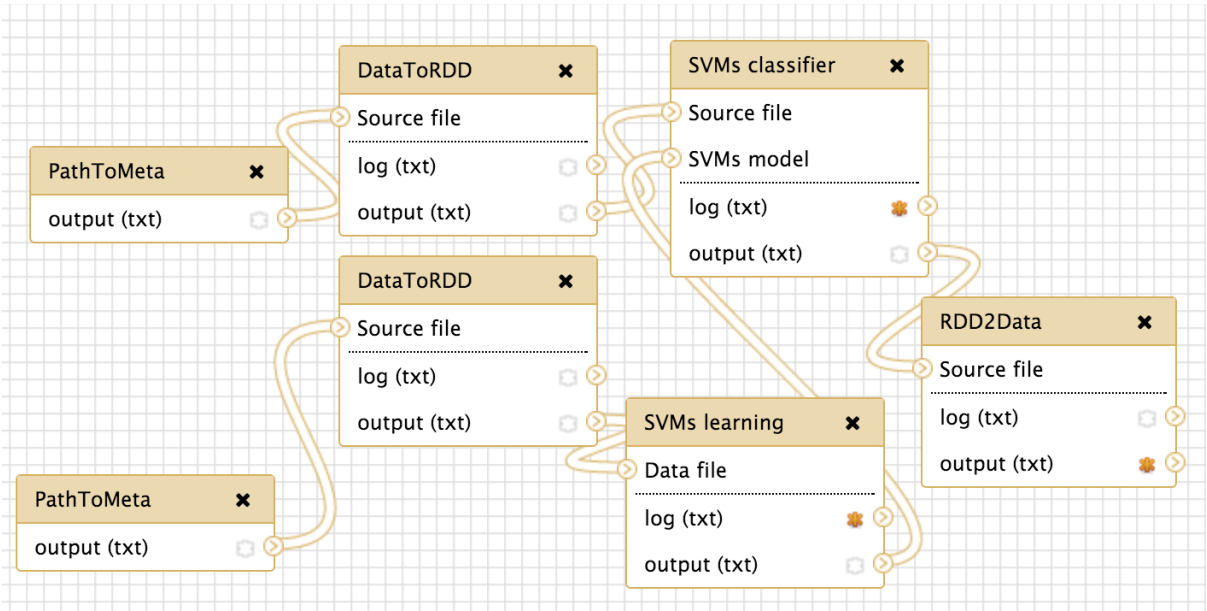

Figure 10: Classification workflow using SVM.

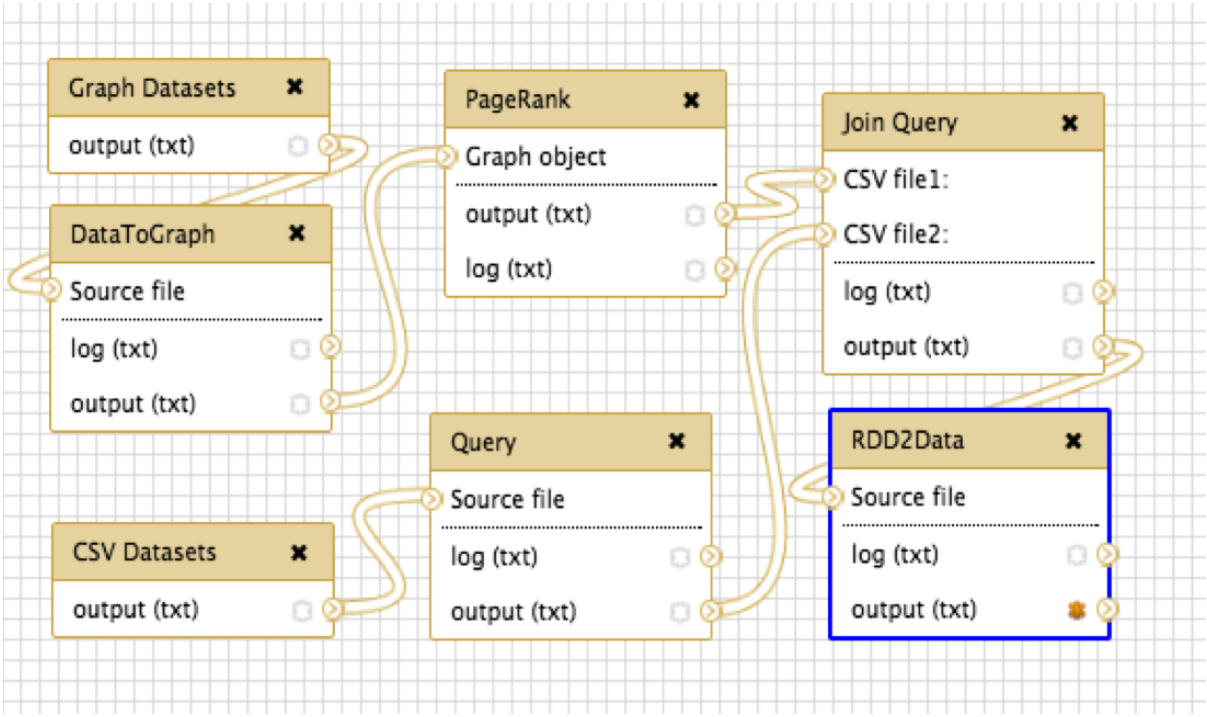

Figure 11: Ranking universities using Wikipedia

4.4 Ranking using Wikipedia

In this workflow, we try to mine the Wikipedia datasets[5]. This dataset is a crowd-source gathered information from Wikipedia and includes several data files such as pagelinks and abstracts. Each line in the pagelink dataset contains a pair of URIs such that the second URI appears in the Wikipedia webpage of the first URI. The abstracts includes the URI of a Wikipedia page and first paragraph of the page. Moreover, the Wikipedia dataset includes the same information for Wikipedia pages in different language. However, we limit our experiment to English pages. Our goal here is to rank universities based on their appearance in the Wikipedia using the PageRank algorithm [10]. Therefore, the rank of a university depends on the Wikipedia pages that have links to the Wikipedia page of the university and importance of those pages based on the ranking.

To find the Wikipedia pages of universities we simply use the URI name and look for related words such university, institute, or college. An alternative approach would using the abstract file, but here the URI name seems sufficient. Therefore the result of the search is a dataframe that includes the id and URI of universities.

To create the graph of Wikipedia, we parse the pagelink data file, and by assigning a unique id to each URI, we convert the pagelink file into an edge-view file. Moreover, we keep the URIs and the assigned ids in a CSV file as URI data file, which we use for detecting university Wikipedia pages.

[5]http://wiki.dbpedia.org/

Table 1: Top 10 universities found using workflow of Figure 11 compared to Wikipedia university ranking from [10].

	Ranking from [10]	SparkGalaxy
1st	University of Cambridge	Harvard Universitiy
2nd	University of Oxford	University of Oxford
3rd	Harvard University	Columbia University
4th	Columbia University	University of Cambridge
5th	Princeton University	Yale University
6th	Massachusetts Institute of Technology	Stanford University
7th	University of Chicago	University of California, Berkeley
8th	Stanford University	Massachusetts Institute of Technology
9th	Yale University	University of Michigan
10th	University of California, Berkeley	Princeton University

Figure 11 shows the workflow of the experiment. The graph dataset points to the edge-view of Wikipedia graph constructed from the pagelink file, and the CSV dataset points to the URI data file. The Query tool recognizes universities in the URI dataframe registered as relational table *uri*:

```
SELECT vertex, name from uri
WHERE (name LIKE '\%University\%'
OR name LIKE '\%Institute\%'
OR name LIKE '\%College\%')
```

The PageRank tool ranks the vertices of the Wikipedia graph, and the output rank dataframe is given to JoinQuery tool. The SQL query given to the JoinQuery joins two dataframes, so the output dataframe only include the rank of universities found using the Query tool. The JoinQuery register the output of the PageRank and Query tools as relational tables *ranks* and *univ*, respectively, and runs the following SQL query on them:

```
SELECT name, rank from univ, ranks
WHERE ranks.vertex = univ.vertex
ORDER BY ranks.rank DESC limit 10
```

Table 1 includes the final ranking as the result of the given workflow as well as the Wikipedia ranking reported by Lages et al. [10]. The set of top 10 universities have 9 intersects. The difference in ranking is mainly attributable to this fact that we only used English Wikipedia pages while Lages et. al use all provided Wikipedia pages.

5 Feature work and conclusion

SparkGalaxy can still be improved by following extensions:

- Galaxy is Web interface has the functionallity of managing the data in the experiment history, so as soon as the user removes a tool result from the history, Galaxy will remove the file from the filesystem. However, because in the SparkGalaxy toolkit, Galaxy only has Metafiles of the real data, SparkGalaxy needs to remove the real data when Galaxy deletes the Metafile. Currently the users can manually delete the intermediate data.

- Galaxy runs the tools separately, so in the current model they cannot share Spark sessions. However, running serveral tools in one session could avoid serialize-deserialize overhead of intermediate data. The solution would be creating customized workflow management system for SparkGalaxy, so we can have more flexibility for executing Spark-based tools. Another application for this customized workflow management is handling stream data. Galaxy cannot runs two tools in parallel if they depends on each other, so it is suitable for piping tools together to process stream data.

- Finally, to boost the application of SparkGalaxy, we need to have a ready to use distributions of SparkGalaxy for different cluster systems such as Amazon EC2.

To recapitulate, we introduced the SparkGalaxy toolkit, which can simplify many big data science experiments. SparkGalaxy provides the user a set of Spark-based tools that can be combined together using Galaxy workflow manger in order to describe complex data science experiments. Researchers can re-run their experiments with different parameter settings and over different input data, and SparkGalaxy encapsulates the complexity of interacting with cluster systems and data-parallel processing.

References

[1] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

[2] Kenny Bastani. What graph analysis of wikipedia tells us about the relevancy of recent knowledge. *http://www.kennybastani.com/2014/12/graph-analysis-wikipedia-recent-relevancy.html*, 2014.

[3] Asa Ben-Hur and Isabelle Guyon. Detecting stable clusters using principal component analysis. *Functional Genomics: Methods and Protocols*, pages 159–182, 2003.

[4] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):172–188, 2008.

[5] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[6] David V Foster, Jacob G Foster, Peter Grassberger, and Maya Paczuski. Clustering drives assortativity and community structure in ensembles of networks. *Physical Review E*, 84(6):066117, 2011.

[7] Jeremy Goecks, Anton Nekrutenko, James Taylor, et al. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.

[8] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 17–30, 2012.

[9] Zan Huang. Link prediction based on graph topology: The predictive value of generalized clustering coefficient. *Available at SSRN 1634014*, 2010.

[10] José Lages, Antoine Patt, and Dima L. Shepelyansky. Wikipedia ranking of world universities. *arXiv:1511.09021 [cs.SI]*, 2015.

[11] Kevin Lang. Finding good nearly balanced cuts in power law graphs. *Technical Report, Yahoo! Research Labs*, 2004.

[12] Frank Lin and William W Cohen. Power iteration clustering. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 655–662, 2010.

[13] Bo Liu, Ravi K Madduri, Borja Sotomayor, Kyle Chard, Lukasz Lacinski, Utpal J Dave, Jianqiang Li, Chunchen Liu, and Ian T Foster. Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses. *Journal of biomedical informatics*, 49:119–133, 2014.

[14] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, 2012.

[15] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[16] Tolga Oztan, Robert Sinkovitz, and Telmo Menezes. Complex social science (cossci) gateway: Autocorrelation modeling, kinship network modeling, k- and pairwise cohesion in large networks open opportunities for online education. http://socscicompute.ss.uci.edu.

[17] Luca Pireddu, Simone Leo, Nicola Soranzo, and Gianluigi Zanetti. A hadoop-galaxy adapter for user-friendly and scalable data-intensive bioinformatics in galaxy. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 184–191. ACM, 2014.

[18] Alessandra Sala, Lili Cao, Christo Wilson, Robert Zablit, Haitao Zheng, and Ben Y Zhao. Measurement-calibrated graph models for social network experiments. In *Proceedings of the 19th international conference on World wide web*, pages 861–870. ACM, 2010.

[19] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[20] Clare Sloggett, Nuwan Goonasekera, and Enis Afgan. Bioblend: automating pipeline analyses within galaxy and cloudman. *Bioinformatics*, 29(13):1685–1686, 2013.

[21] Daniel A Spielmat and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 96–105. IEEE, 1996.

[22] Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 2:1–2:6, New York, NY, USA, 2013.

[23] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1029–1040. ACM, 2007.

[24] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.