Big Data Analytics Toolkit for Business Data Based on Social Network Analysis

by

Fan Liang

Master of Software Engineering, Southeast University, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Weichang Du, Ph.D., Computer Science
Examining Board:	Suprio Ray, Ph.D., Computer Science
	Huajie Zhang, Ph.D., Computer Science
	Donglei Du, Ph.D., Business Administration

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2016

©Fan Liang, 2016

Abstract

Social network analysis (SNA) measures the relationships and structures with a set of metrics by building graphs for capturing influential actors and patterns. In this thesis, we investigate the SNA approaches for solving real-world business applications, and propose a general-purpose software system that combines big data analytics and social network analysis techniques. The system's workflow consists of data collection, graph generation, graph reuse, network property calculation, SNA result interpretation, and application integration. The system operations are executable in a Hadoop-based distributed cluster with high throughput on large-scale data. We evaluate our prototype system with a case study on stock network. The result shows that the system is capable of analyzing business data at-scale and using SNA approach to solve business problems.

Dedication

I would like to dedicate this thesis work to my parents for their supports for my dream.

Acknowledgements

I would like to sincerely thank my supervisor, Dr. Weichang Du, for his guidance and support throughout my study. He were always willing to offer me help and answered my questions with patience. I did learn a lot from him.

I would also like to thank all the staffs of Faculty of Computer Science, University of New Brunswick for offering me guides and facilities throughout my academic years.

To all my friends, thank you for your companion, sharing and encouragements. I appreciate the moments we spent together.

Table of Contents

Abstract ii

Dedication iii

Acknowledgments iv

Table of Contents vii

List of Tables ix

List of Figures x

1 Introduction 1

1.1 Motivation 1

1.2 Research Objective 2

1.3 System Overview 4

1.4 Thesis Structure 5

2 Background and Related Work 6

2.1 Social Network Analysis 6

2.2 Hadoop 8

2.3 Graph Database 11

2.4 Business Applications Using SNA 14

2.5 Related Work and Comparison 19

3 System Design **21**

3.1 System Features 21

3.2 System Architecture 24

4 System Implementation Framework **28**

4.1 Data Collection 28

4.1.1 General Process 28

4.1.2 Data Format 29

4.2 Graph Generation 31

4.2.1 Graph Generation Specification 32

4.2.2 Graph Generating Process 34

4.3 Graph Reuse 36

4.3.1 Related Terms 36

4.3.2 General Process 37

4.3.3 Virtual Graph Storage and Transformation 39

4.4 SNA Property Searching 42

4.4.1 Property List 44

4.4.2 Property Calculation 44

4.4.3 Property Result 45

4.4.4 Viewing Property 46

4.5 Business Application Interpretation 46

4.5.1 Result File Hierarchy 46

4.5.2 Interpretation Formats 47

5 Prototype System Implementation **49**

5.1 Data Collection 51

5.2 Graph Generation 53

5.3 Graph Reuse 58

5.4 SNA Property Searching 61
5.5 Business Application Interpretation 64

6 Case Study **67**
6.1 Data Preparation and Collection 68
6.2 Network Construction 69
6.3 SNA Property Calculation 70
6.4 Result Retrieving and Interpretation 72
6.5 Network Comparison 72

7 Experiments **74**
7.1 Experimental Environment 74
7.2 Experimental Result 75
7.2.1 Centrality Measures 75
7.2.2 Network Comparisons 79

8 Conclusion **82**
8.1 Summary 82
8.2 Contribution 83
8.3 Future Work 84

Bibliography **90**

Vita

List of Tables

2.1 Graph Models in Business Applications 19

4.1 Time Sequence Format 30
4.2 Data Unit Table 30
4.3 Time Series Table 30
4.4 Transction List Format 31
4.5 Common Attributes in Graph Generation Specification 33
4.6 Attributes for Time Sequence Specification 33
4.7 Attributes for Transaction Specification 33
4.8 Graph Reuse for Time Sequence Format 43
4.9 Graph Reuse for Transaction List Format 43
4.10 Classification of Property Result 45
4.11 Classification of Result Interpreting Formats 47

5.1 Data Unit Table Schema 52
5.2 Time Series Table Schema 52

6.1 Example for Graph Generation Specification 71

7.1 Matrix File 75
7.2 The Top Five Stocks with the Higher Centralties 77
7.3 Centrality Scores 78
7.4 Sample Time Periods 79
7.5 The Top Ranks Over Some Periods 81

List of Figures

2.1 HDFS Architecture 9
2.2 Yarn Architecture 11
2.3 The Property Graph of Friends 12

3.1 System Architecture 25
3.2 System Progress 26

4.1 Data Collection Process 28
4.2 Graph Generation Context 32
4.3 Building Graph for Time Sequence Format 34
4.4 MapReduce Jobs 35
4.5 Graph Reuse Context 38
4.6 Property Searching Process 42
4.7 File Hierarchy 47

5.1 The Physical Structure of the Repository 50
5.2 The Class Diagram of the Source Parser Component 53
5.3 The Class Diagram of General Graph Generation 54
5.4 The Class Diagram of Graph Loader Component for Time Seuqence Format 56
5.5 The Class Diagram of Virtual Graph Saving 59
5.6 The Class Diagram of Virtual Graph Generation 60
5.7 The Class Diagram of Property Searching 63

5.8 The Class Diagram of Eigenvector Centrality Calculation 64
5.9 The Class Diagram of Property Result Interpreter Component 65

6.1 An example of Stock File from Yahoo Finance 69
6.2 An example of the Time Sequence File 69

7.1 The Stock Network Structrue of the Graph in Period 1 78
7.2 Evolution of the Numbers of Nodes and Edges 80

Chapter 1

Introduction

1.1 Motivation

Big data is a hot spot in recent years. It refers to the fact that the datasets are so large and complex that the traditional applications are not capable of processing the data [3]. Now more companies believe that if they can capture all the data and apply analytics, it can unlock significant value, uncover patterns and trends which lead to better quality of products and better decisions. For example, the US health care system could create more than 300 billion dollars in value every year if it were to use big data effectively to drive efficiency and quality [4]. To understand the underlying patterns, identify market trends, spot weakness and make more informed decisions, big data analytics is to be developed for the process of examining big data by using advanced analytic techniques [5]. Today large portions of data are semi-structured or unstructured including web logs, social media, business activities reports and clickstreams. It is not very appropriate to store these types of data in traditional data warehouses based on relational databases. In contrast, new techniques in big data analytics like Hadoop, Spark and Hive are capable of processing large volume of data with a broad variety of types.

Social network analysis (SNA) studies the interactions among the actors under the assumptions that the communications of the individuals have an impact on the groups and patterns by using the tools of graph theory, statistics and computer technologies [6]. To understand the function of an individual in a group, it is more effective to measure the person's structural relations quantitatively than the person's attributes (e.g. age, gender, role). Moreover, how the network was formed and its structural characteristics have a strong influence on performance of the network. A SNA perspective provides a set of metrics for analyzing networks, such as density, diameter and clustering coefficient etc.

To date, the social network analysis approach is widely applied not only in sociology, but also in politics, economics, biology, world wide web and geography. In business analytics, business networks capture the interactions between business entitites. Taking advantage of the techniques of social network analysis, researchers discover the underlying processes and patterns in business networks, which lead to making strategic decisions more efficiently and higher profit. Combining big data with SNA in business, data scientists can build a large graph and mine it, which provides new methods to discover market trends and consumer sentiment [7].

1.2 Research Objective

The objective of the research is to develop a general-purpose software system to analyze large business datasets using SNA approach implemented on Hadoop. The analysis includes several phases including network generation, property searching and result interpretation. The system helps users to build their graph models and analyze the graphs with SNA algorithms. The output results can be integrated with the user applications. The system is built on Hadoop which enables users to store, create and process large-scale distributed graphs. The target users of the system are

not required to have much knowledge of graph processing and big data techniques. By using the system, users can handle large datasets with SNA more effectively and efficiently.

To achieve the system's goals, there are some general research problems we need to solve. The first problem is how to generate graphs from source data. For different research purposes, the methods to build graph model are very different. Although the graph model has been decided by the user, the graph building process can be diverse. Besides, the formats of source data are heterogeneous, as they are usually collected from various channels such as from surveys, questionnaires, interviews, the internet, or enterprise databases. The formats of the data can be tables, emails and hyperlinks. The key is how to identify nodes and edges and extract them from the source data.

The second problem is how to store graphs in the system. It may be a large graph. Besides, it should support graph query during property searching. The traditional SQL database is not suitable for storing graphs. We may store the graph data on the distributed file system, and leverage some distributed software and engines to manage the data. The key lies in what forms of data should be stored for graph searching efficiency.

The third problem is how to interpret the property results for various formats and semantic contexts. The output forms of the property results are different. For example, the density of the graph is a single value, while the clique is a subset of the graph consisting of a set of nodes. The results are used for different analytic purposes. Combining with other results, users may want to use the property results such as centrality to calculate correlations. In this case, the results are vectors. In another case users may want to store the results in graphs for further pattern discoveries. For example, the clique results are used to create a new graph for property searching. The key to solve this problem lies in what formats of results to

interpret for application integration.

1.3 System Overview

The proposed system is capable of handling large datasets and it is suitable for varieties of business purposes with varieties of SNA algorithms. It consists of a pipeline work including data collection, graph generation, graph reuse, property searching and business application interpretation.

The system extracts and transforms source data into graphs automatically. Actors are nodes and edges are the correlations and activities in business domain. We conducted a survey by reading research papers and articles which applied SNA to business data. We investigated how the researchers built their graph models from the source data. By analyzing the merits of the graph models, we define several data formats in the system. Each data format underlies a certain graph building method. Data described in these papers can be effectively converted to one of the formats, and the system can read such formatted data to generate graphs.

The system has a repository to store all the data, including source data, graphs, and property results. We use graph database to help us manage graph data in the repository. It is a NoSQL database with emphasis on relationships concept. Data stored in the graph database are in graph structure naturally which can be quickly traversed and searched. It is scalable on large datasets and suitable to our needs.

After getting the SNA graph, the graph analysis work starts which focuses on calculating properties of graph by using social network analysis. The system facilitates some basic property calculations. With the graph properties, the system offers flexible methods to interpret the results to business applications. To decide what formats of results should be produced for interpretations, we also refer to the business research papers and applications. The system provides facilities to support common

SNA property usage for business applications. The system uses Hadoop as the base platform to store and process large datasets in a distributed cluster.

1.4 Thesis Structure

The rest of the thesis is organized as follows: Chapter 2 introduces the background and some related work. Chapter 3 describes the system design with the general requirements, system features and system architecture. Chapter 4 describes the realization of the system with the five major components: data collection, graph generation, graph reuse, property searching and business application interpretation. Chapter 5 describes the implementation details of the prototype system. Chapter 6 explains a case study to show how to use the system to analyze stock business data with SNA. Chapter 7 shows the experimental results of the case study and other experiments using the prototype system. Chapter 8 gives the conclusion and future work.

Chapter 2

Background and Related Work

2.1 Social Network Analysis

Social network (SN) is a social structure consisting of a set of social actors and a set of dyadic ties between the actors [8]. The actors are also described as nodes and the interactions and relationships between actors are edges which link nodes together. The structure of the networks is analyzed to identify influential actors and patterns with a set of methods. Social network analysis (SNA) studies these structures. There are many definitions of SNA. One definition is that "Social network analysis [SNA] is the mapping and measuring of relationships and flows between people, groups, organizations, computers, URLs, and other connected information/knowledge entities." [9].

The SNA approach combines technical and mathematical methodologies as an interdisciplinary filed including social psychology, sociology, statistics and graph theory [8]. By using these methods, many interesting related theories have been discovered about human society, such as small world phenomenon, six degree of separation, the friendship paradox, the strength of weak ties, giant component, etc [9] [10]. The SNA approach is now not only used in sociology but also used in a wide range of

domains, such as politics, economics, biology, world wide web and geography. Using SNA approach, a series of networks are defined and built in various scenarios, such as social media networks, collaboration networks, disease networks, and business networks. In the business network, nodes can be firms, shareholders, products, customers, and edges can be activity links, resource ties and actor bonds [11].

A network can be viewed as a graph. SNA categorizes and presents graphs in terms of the types of relationships such as directed or undirected, weighted or unweighted, simple or multi graphs (lops or multiedges). There are numerous SNA metrics and algorithms to examine and measure the characteristics of graphs. The graph properties reveal the network structures, the strength of the relationships and roles of the actors. The SNA properties can be divided into two groups from the perspective of the actor and the network [11]. In the actor perspective, to identify key players in networks, researchers use "centrality" as network properties measurements, including degree centrality, closeness centrality, betweenness centrality, eigenvector centrality and PageRank. Degree centrality captures the node with the most connections to others. Closeness of a node is the inverse of the farness which is the sum of its distance to other nodes. Closeness centrality measures the node having the lowest total distance to other nodes. Betweenness centrality measures the number of times a node lying in the shortest path of other node pairs. Both eigenvector centrality and PageRank depend on the number of neighbouring nodes and how importance of the neighbours, while eigenvector centrality is for undirected graph and PageRank is for directed network. PageRank is initially used in Google to rank the relevant web pages in web search. In the network perspective, there are a set of metrics to measure the entire network, including density, distance, diameter, mean path length, degree distribution and homophily. Density measures the number of edges existed in the graph compared to the maximum possible number of edges [12]. The distance of two nodes is the shortest path of edges connecting the two nodes. The diameter

of a graph is the largest distance of any two nodes, and the mean path length is the average distance of all the node pairs. Degree distribution is the probability distribution of the occurrence of degree over the network. Homophily measures the tendency of the nodes to link together that are similar.

2.2 Hadoop

Hadoop is a framework for distributed storage and processing of large datasets on clusters [13]. Hadoop is open source featuring low costs, flexibility and scalability. It outperforms when processing terabytes and petabytes of data sets with commodity hardware. It is robust in parallel calculation and can run on thousands of nodes smoothly. The core of Hadoop consists of two parts: the Hadoop Distributed File System (HDFS) and MapReduce. The HDFS is for distributed storage and MapReduce is for distributed processing. HDFS splits large data into blocks and distributes them across the cluster. When processing the data, the MapReduce copies the code to each node in the cluster and processes them in parallel. Each node does its local task. A MapReduce job synchronizes the tasks. This framework allows us to handle sheer volume of data more efficiently.

Hadoop was created by Doug Cutting, who was inspired by Google's distributed system which was published in Google's paper in 2003 [14] [15]. It has become an independent project in Apache in 2006. The Apache Hadoop is an open source project composed of four modules: Hadoop Common, Hadoop Distirbuted File System (HDFS), Hadoop Yarn and Hadoop MapReduce [13]. There are also vendors working on offering enterprise solutions based on Hadoop, such as Cloudera, Hortonworks and MapR. Next, we describe the Hadoop architecture based on Apache Hadoop.

HDFS is a distributed file system which stores large files in commodity servers as

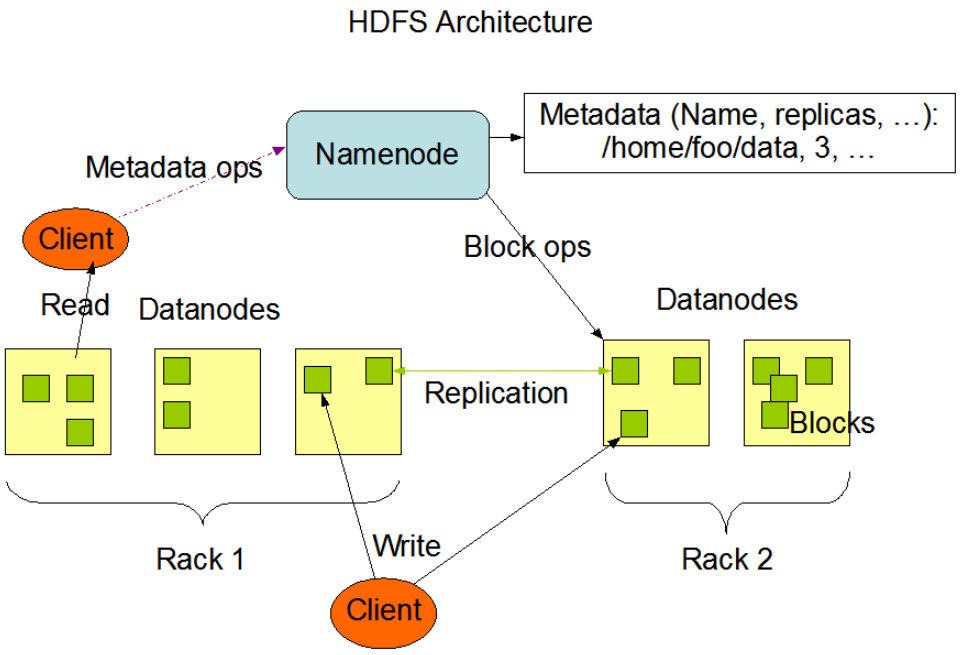


Figure 2.1: HDFS Architecture

high-throughput data access by streaming [13]. Data in HDFS are read and written in blocks. A block is an independent unit in a fixed size of 64MB. A HDFS cluster consists of two types of nodes: namenode and datanode [1]. Namenode maintains bookkeeping information about the file system tree and metadata, while datanode stores data and informs the data lists to namenode. HDFS is designed for a write-once, read-many-times pattern [14]. During writing process, files are replicas in chosen datanodes being aware of making balance between reliability and bandwidth. When reading files, data is retrieved by clients directly from datanodes selected by namenode for the best ones. Figure 2.1 shows the HDFS architecture [1].

MapReduce is a parallel data processing system by splitting jobs into tasks running independently. When reading large-scale data distributed over a number of machines, the traditional relational database system is not the best solution anymore, because seeking and moving data units among multiple disks may take longer than through streaming which is related to transfer rate. MapReduce is very efficient in handling

batch analysis, especially with semi-structured and unstructured data [14]. Data in MapReduce are analyzed by user-defined map and reduce functions. Both take key-value pairs as their input and output. In map phase, data locality optimization is applied as intermediated data are read and written locally to avoid too much traffic load in clusters. Then reduce task will take the output of map as its input in HDFS. When there are multiple reducers, the output of mapper is partitioned and flowed to different reducers, known as shuffle, which is an essential contribution to MapReduce performance. The final output of reduce function will be written to the HDFS directory and can also be written to the local file system. The classic MapReduce Framework in Hadoop 1.x versions consists of a jobtracker and tasktrackers.

In Hadoop 2, a new framework called YARN (also called MapReuce 2.0) replaces JobTracker and TaskTracker with ResourceManager and NodeManager. The ResourceManager is composed by two components: Scheduler and ApplicationManager. The Scheduler is responsible for scheduling jobs, and the ApplicationManager is for monitoring task progress. NodeManagers are installed on machines individually and run local tasks in containers meanwhile keeping heartbeats with the ResourceManager. Compared to the classic MapReduce, MapReduce 2.0 is more compatible and flexible. It allows different applications to excute in the same cluster and supports different MapReduce versions. Users can configure the ranges of memory size used by the tasks which is a fixed value in the classic MapReduce. Figure 2.2 shows the YARN architecture [2].

Today the Hadoop ecosystem involves many other projects cooperating for large data computing, such as Cassandra, HBase, Hive, Pig, Spark, ZooKeeper. Many large companies take advantages of Hadoop based technology, such as Yahoo, Facebook, Twitter, and the New York Times [16].

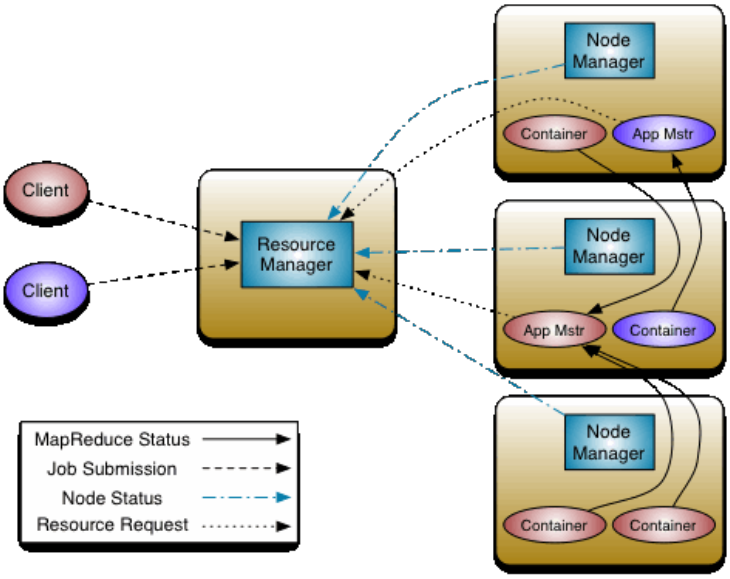


Figure 2.2: Yarn Architecture

2.3 Graph Database

Graph databases store data in graph structures for nodes and their edges in the networks. It is one type of NoSQL databases, and it is useful for analyzing interconnections, and it is even more expressive than relational database.

Graph databases can be categorized into property graph, hypergraph and triples. A property graph consists of nodes and edges which describe the relationships between nodes. Both nodes and edges have distinguished properties. A hypergraph can use edges to connect arbitrary number of nodes. A property graph is capable of replacing a hypergraph with multiple edges. Triples state in a "subject-predicate-object" form comes from semantic web. An URI RDF triple represents in URIs and each unique URL associates to an element in the triple. Triples support high-scalability by storing in several documents spreading across servers. Property graph is the most popular graph database in the three types for it is simple and easy to understand and captures sophisticated domains from government to business. In a property graph, nodes represent entities such as individuals or business. Properties

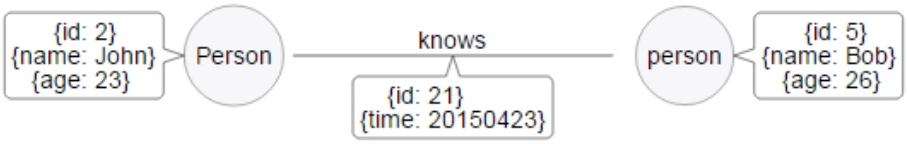


Figure 2.3: The Property Graph of Friends

are relevant attributes relating to the nodes. For example, a node representing a person "John" may have properties like name and age depending on which aspects of that person are wanted to be involved in the database. Edges are connections between nodes which also have properties. For example, when two people "John" and "Bob" become friends in the Facebook, an edge can be drawn to connect them with the property "knows" as its label indicating that they know each other in the Facebook. Figure 2.3 shows an example property graph.

The graph database supports CRUD (create, read, update and delete) operations and OLTP (online transaction processing) based on graph structure. In the storage backend, graph databases use different physical layers to serialize the data ranging from a relational database to an object-oriented database. The existing graph database projects are different in terms of their physical techniques and architectures. Neo4j is one of the world's leading graph database. It supports fast data query at scale. It uses native graph storage in graph management and has no index, as data are stored in adjacency matrix pointing to their neighbours physically [17]. It can leverage Amazon AWS cloud to store and access scaled graphs. Titan is a distributed graph database. It supports three storage backends: Oracle BerkelyDB, Cassandra and HBase. It is capable of storing and querying graphs with billions of vertices and edges [18].

Graph analysis frameworks offer solutions for analyzing distributed graphs. Google's Pregel is inspired by the Bulk Synchronous Parallel (BSP) model of distributed

computation [19]. Apache Giraph is derived from Pregel built on Hadoop. It is an iterative graph processing framework built for high scalability [20]. With the framework, users can write their algorithms and run Giraph-based jobs for graph computations. GraphX is another Apache project based on Spark for graph-parallel computation. It supports graph building, graph operations and has a library of graph algorithms [21]. Faunus is a graph analytic engine based on Hadoop which uses MapReduce for graph computation. It is now called Titan-Hadoop and is one of TinkerPop 3's packages. It also integrates with Giraph and Spark for graph OLAP [22].

Graph databases have some priorities in adapting business models in practice for its powerful performance, quick business reaction and developmental agility. Unlike the join pains and declining performance in relational databases, graph databases can respond stable to end users in milliseconds for certain query tasks. Moreover, graph databases have dynamic schema which adapt well to the fast changing business. In the past, when a new requirement came from users, it might have been taken a long period for modifying schema, transferring and migrating data to a new structure. Now the flexibility of graph database enables to evolve its schema as business model evolves by adding new nodes and relations to the existing graph. Last but not the least, graph databases shorten the period of each iterative and incremental agile cycle. In traditional design, developers usually make a lot of efforts in transforming and testing data to fill the gap between the logic design and relational databases. Graph databases are pragmatic and expressive in software development projects. Large companies like LinkedIn, Wal-mart and eBay are using graph databases in their business [23].

2.4 Business Applications Using SNA

To develop the system, we investigated how SNA approach is used in business applications to solve real-world problems by reading 21 business research papers and articles published in the last 6 years. In each paper, the authors describe their solutions of building networks and apply SNA to analyze the networks. Combining the knowledge of the business domain, the SNA results are usually interpreted to some semantic meanings for further plausible results. The applications in the papers cover a range of business areas including supply chain, e-commerce, blogosphere, business process analysis, inter-company network, financial market,f market basket analysis, stocks, mobile phone, insurance and social media. We focus on the three issues on each paper: 1. how the graph model was built, 2. what SNA properties were used to analyze the graph(s), 3. what methodologies were used and what the results were. We summarized these three issues in the 21 business research papers below.

In supply chain, SNA is used to analyze the role-players in supply network. Kim et al. [24] provided a framework to build and analyze supply networks. Companies in the supply chains were nodes. Three supply networks were built in terms of three types of links, which were material flows, contractual relationships and product lines. The SNA properties used to measure the networks were degree centrality, closeness centrality, betweenness centrality, density, network centralization and network complexity. The property results were compared with case-based interpretations in Choi&Hong(2002) [25]. The results showed that the framework can both supplement and complement to the case-base analysis. Marino et al. [26] built supply networks for the Italian beef market. Nodes in the networks were the agents in the market. Edges were created when the agents appear in the same tag's the animal's ear, which presented their market exchange. The SNA properties used in the networks were density, distance, degree centrality, closeness centrality, betweenness centrality, hub and authority scores. The paper also defined three properties: coordinate u,

coordinate v and SCA, which measure the number of suppliers in competition, the number of buyers in competition and number of role-players removed which would disconnect the group. The paper applied SNA to the beef market to understand the interrelationships of the agriculture-food supply chain. One of the conclusions was that more strong farmers are needed to promote the beef selection process.

In World Wide Web, networks are built with hyperlinks. Stephen et al. [27] and Fieseler et al. [28] both analyzed the actors on the Internet. Stephen et al. [27] studied social commerce on online shops. The online shops were nodes and the hyperlinks between the shops were edges. The properties are degree centrality, proximity, closeness centrality, hub centrality and authority centrality. Comparing the SNA results with the profits, the paper drawed some conclusions: shops with higher centrality were more accessible; the social commerce connection contributed to economics value; the most benefit seller was the most accessed one, not the most central one. Fieseler et al. [28] built an ego-network on McDonald's CSR (corporate social responsibility) blog. Nodes were the blogs and edges were the hyperlinks between blogs. The properties used were density and degree centrality. The results showed that the stakeholder involvements via blogs were valuable.

SNA is also used to analyze activities inside companies. To manage information and optimize business process, Levina et al. [29] built two networks. The first network was for business process analysis. Nodes in the network were business process activities and edges were information flows between the activities. The second one was a content-based clustering network where the edges were sharing content objects such as reports and presentations. The properties used were average path length, closeness centrality, betweenness centrality, degree centrality, clustering. The paper maped the structure of the contented clustering network to the process network to derive knowledge and information structures of the process changes. Palus et al. [30] built an email network using email accounts as nodes and emails as edges. It defined a social

score to measure each node in the network, which combined a series of properties, such as degree, degree centrality, closeness centrality, betweenness centrality, average shortest path, clique scores. A hierarchical position (HP) score was also calculated according to the positions of the workers in the company. The author used Kendall's ranking comparison method to calculate correlation coefficient between HP scores and social scores. The results were used to manage human resources.

Business network analysis builds inter-company networks to evaluate innovations and knowledge in the industry. Gloor et al. [31] analyzed the LinkedIn, Facebook and email networks of the startup entrepreneurs in the Swissnex community. The authors used degree centrality and core/periphery to measure the networks. They also calculated a success score for each entrepreneurs and academics by their professional positions. They compared the success of startup entrepreneurs and innovations with their SNA properties. They concluded that the centrality could predict entrepreneurial and academic success. Giuliani et al. [32] used SNA for evaluating cluster development programs (CDP) outputs and impacts on host regions and populations. The authors built two types of networks. One was "formal" and one was "informal". Nodes were different types of actors, including companies, universities, government and nonprofit organizations. Edges in the "formal" network represented trade of goods and contractual agreements between actors, while edges in the "informal" network represented transferring or tacit knowledge. The properties used in the networks included degree centrality, betweenness centrality, bonacich centrality (also called eigenvector centrality), clustering coefficient, degree distributiaon, and clique. To learn changes of the cluster firms, the authors compared the property values proceed time by time (prior, during, after the policy treatment) to measure significant changes. Egeraat et al. [33] built knowledge and innovation networks in the Irish biotech industry and learns the structural characteristics. The authors built "informal" and "formal" networks. Edges in the "informal" network

represented directorship while edges in the "formal" network represented patent and research collaboration. The properties measured were average path length, average clustering coefficient and density. The results showed that the networks were small networks with little collaboration, and the formal network was less clustering than the informal network. Fuks er al. [11] investigated the network relationships of the members of electronic freight exchange called Trans.eu. The nodes in the network were companies. The edges were created when the transactions are over ten times. The measuring properties were density, clustering coefficient, small world effect, average degree, diameter, eigenvector, closeness and betweenness centrality. Miller [34] examined the network structure of the film companies in the U.S. market. Nodes in the networks were film companies and edges presented their coproduction. The author built two networks for high grossing films and highly lauded films and compares them. The measuring properties were degree centrality, clustering, density and homophily. The results concluded that future film production looks more cooperative, dispersed and international in structure.

Rancan [35] and Siregar et al. Rancan [36] measured financial market from the SNA perspective. [35] aimed to evaluate the values of social network in financial market. It built a network in which nodes were fund managers and edges represent whether the fund managers had attended the same institution and gained the same degree. The properties used weree density, shortest path, average path length, degree and closeness centrality. The authors investigated mutual funds performance and associated with the network structure. The results were that larger funds hire well-connected managers, and better socially connected managers have a better performance. Siregar et al. [36] studied the relationships of structural positions and financial performances of APEX-Rural banks in Central Java network (CJ-Net). Nodes were banks in the CJ-Net and edges represented consultation and discussion between the bank members. The author used degree centrality, closeness centrality

and structural holes to measure the network. By calculating correlations between their structural positions and financial performances, the authors found that strong structural positions have better financial performances.

To analyze and predict the stock market, stock networks can be built on stock price correlation. Roy et al. [37] built stock networks with nodes for stock indices. The authors used degree, betweenness closeness and eigenvector centrality to identify influential stock indices in global market and regional influence on co-movement. Lee et al. [38] built a stock network based on the stocks in the US market. By calculating the four centralities, the authors determined influential stocks in the market.

Market basket analysis learns the customer purchase behaviors. Products co-purchased are linked together, and form a weighted product network. Reader et al. [39] and Videla-Cavieres et al. [40] built product networks using the transaction times as threshold. The SNA properties used in the papers were degree distribution, density, diameter, average clustering coefficient, and community detection.

Cellular networks in mobile services can help telecom operators for recommendation services. In [41], the network was built in which each node represented a customer and links were their SMS transactions. To design better service plans, telecom operators studied social behavior of their customers by calculating degree distribution and identifying components in the network.

In insurance industry, SNA can be used to detect suspicious participants. Pinheiro [42] described a network in which participants who were in one claim were connected. The authors calculated the properties distinctively in terms of the participants' roles. The properties used were community detection, degree, closeness, betweenness and eigenvector centrality.

In Twitter and microblog applications, SNA is used to magnify marketing effects in social network. Zeng et al. [43] used centrality properties to identifying influential users. Identifying influential tweets and tagging key hashtags in the tweets can

also be used in marketing or advertisements. Hashtagify.me is an advanced search engine which contructs hashtag networks when the hashtags appeared in the same tweets [44].

We summarized building graph models for the above business applications in Table 2.1. Note that this list only shows some main methods for graph building. It does not mean that these are the only graph models, but it does give a glimpse of common graph models used in business applications.

Table 2.1: Graph Models in Business Applications

Business Areas	Relationships	Actors
Supply chain	Material exchanges/flows	Agents/firms
Online shops, blogosphere	hyperlinks	Shops, blogs
Business process analysis, company analysis	Infomation flows/exchanges	Process, email accounts
Business network analysis	Email exchanges, trade/-transactions, directorship, consultation/discussion, contractorship, coproductions	Firms, universities, nonprofit organizations, government agencies, banks
Market basket analysis	Products bought in a transaction	Products
Stock market	Similarities associating with stocks price	Stocks, stock indices
Mobile service	Message exchange	Customers
Insurance	Participants who are in one claim	Participants
Twitter/microblogs	Hashtags appear in the same tweets	Hashtags, accounts

2.5 Related Work and Comparison

Our system is inspired by Junyan Zhang's previous work [45] in which an online toolkit is developed to analyze stock data using the SNA approach. The toolkit builds stock network and uses subgraph selection methods to simulate index trend

and make up investment portfolio. As our work also uses SNA approach for business data analytics, our system is quite different from Zhang's work, and we summarize two main differences.

First, the goals of the systems are different. Zhang's toolkit aims to provide an innovative method to analyze stocks using SNA. The process is building stock network, filtering the subgraph, tracing stock price and making comparison. It is a very specific application scenario for stocks. Compared to his system, our goal is to provide a general-purpose toolkit for analyzing business data in various scenarios. Although our system also includes the process of graph building and analysis, the graph generation method and analytics approaches are varied in terms of different applications. Our system is not supposed to provide new SNA algorithms to analyze business data, but help users apply standard SNA algorithms to their applications by using the system.

Second, Zhang's toolkit is designed not for big data and does not use any big data analytics technique, while our system does. In Zhang's toolkit, the stock data are stored in MySQL in a single machine, and the graph elements are also stored in MySQL as well. R is used for graph processing. In contrast, big data analytics is one of the features of our system. Our system stores and analyzes the data using Hadoop, which is capable of storing and processing large datasets across clusters. The graphs are stored in the graph database which uses graph structure such as nodes and edges to store data. As graph database supports natural graph structure storage, more complex graph queries can be executed on the system.

Chapter 3

System Design

3.1 System Features

The system provides services for building, storing and analyzing SNA graphs based on Hadoop on business data. In this section, we describe the system features in details.

Large data storage and access: The system has a repository to receive and handle sheer volume of data. The data storage should be scalable and can be accessed across clusters. Moreover, data in the repository are in various structures including source data, graphs, SNA properties as well as all kinds of related configurations. Some types of data can be of large volumes while some are not. The system manages to access them for efficient searches.

Data collection: The system takes source files submitted by users and store to its repository. When users upload source data files, the system reads the datasets from the local machine, and reformats the data in the repository for effective graph generation and searching. The source files should be in a limited number of formats that the system can decipher. The current system has built-in methods to read datasets in two specific formats: time sequence format and transaction list format.

Details are provided in the next chapter.

Graph generation: The system builds social networks from users' submitted datasets. By giving specifications, users tells the system the graph type (directed or undirected), how to identify nodes from the datasets and how to calculate weights of edges between pairs of nodes. The system offers limited ways to determine nodes and edges. For each data format, the system defines methods to identify nodes and edges and calculate weights. Users also give what nodes and edges are going to be selected to a graph in the specification. Then the system automates the process to create nodes from the selection and create edges whose weights are within the given interval(s) in the graph database. The system also supports user-defined selection algorithms. Based on the same source data, the user can build various graphs by specifying different selection parameters and methods. The process runs in parallel for high throughput.

Graph reuse: The graph generation allows to generate graphs from source data directly. However, we can also leverage existing graphs and rewrite them to derive new graphs. Consider the situation in which the user wants to do SNA property search on several similar graphs. These graphs may be similar, such as just having more than one or two nodes than the other. It may be inefficient to always generate new graphs from source data which are very much the same, and it is also waste of space to store a group of large graphs that are very alike. Therefore, when users want to do property searching among similar graphs, the system can reuse the existing graphs in the graph database. The general idea is that, instead of generating from data source directly, the system gets new graphs by transforming from the existing graphs.

After the user tells the system how to generate a new graph, the system searches the graph database to find if there is a graph candidate which can transform to the potential new graph. If there is, the system will use the candidate to rewrite a new

graph. The system saves one graph in the graph database. When the user wants a new graph which is identified as similar to that one, the system stores a virtual graph by saving their relationships and their transforming steps between the existing one and the virtual one. The new graph can be a subgraph, a modified graph or an extended graph of the original graph. The system identifies the relationships between the new graph and the original graph according to graph generating specifications. After the relationship is decided, the system should know how to mutate the original graph to the new graph. To guarantee that the output of the new graph is unique, the system predefines certain transforming approaches based on the data format.

We assume that users have little knowledge about graph transformation. They are not aware of whether the new graph comes from directly generating process or from reusing process. The system decides whether to reuse graphs and do the graph transformation work. The process should be automatic and transparent to users.

SNA property searching: When the system is calculating SNA property, it gets rid of the semantic meanings of the nodes and edges. It takes SNA graphs whose attributes of nodes and edges are ignored except weights of edges. The system offers a known list of SNA properties users can search. Users tell what properties to search. The system calculates and searches for the selected properties and shows the results to users. The system implements a list of built-in property programs that users can use. For weighted or unweighted graphs and also for directed or undirected graphs, the system should use different SNA property algorithms for analysis. After the property calculation is completed, the property results are saved to the repository for application interpretations.

SNA property interpretation: For a SNA property, the system provides limited ways to interpret property results for different kinds of applications. Users tell what kind of scenarios for result interpretation. The system generates the results in certain forms and provides them for users to access, or the system returns information telling

how to get the results. For example, for clustering, users can choose to see the results as a list or in the graph. If it is the first scenario, the system can output a list of vertices and provides built-in APIs to access the list. If it is the second scenario, the system writes the cluster value to each vertex in the graph and tells users the graph location. Users can access the results through the graph APIs later.

Moreover, the system provides APIs for users to retrieve SNA property results in two ways: online and offline. Online result retrieving is that users access property results from the repository directly. The system also supports reading results offline, which provides users the result files containing all the results, so that users do not need to connect to the repository every time to get the results.

Graph visualization: After a graph has been generated in the graph database, users can visualize the graph which gives them an intuitive view about what its structure looks like. However, for a large graph, it usually makes no sense for viewing the graph as there are too many nodes and edges knitting together. Therefore, graph visualization is optional in the system.

3.2 System Architecture

The system supports ingestion, processing, and display of graphs on large datasets. It provides a flexible, component based, pipeline architecture for the integration and deployment of SNA property algorithms in the fields of business and social network analysis.

Figure 3.1 shows the system architecture. The diagram shows the data flow inside and outside the system. We assume that users have already had formatted data ready in their local machines. Users input the source files to the system. The data go through a series of graph building and network analysis processes. Finally the system outputs property searching results for users to use in their own applications.

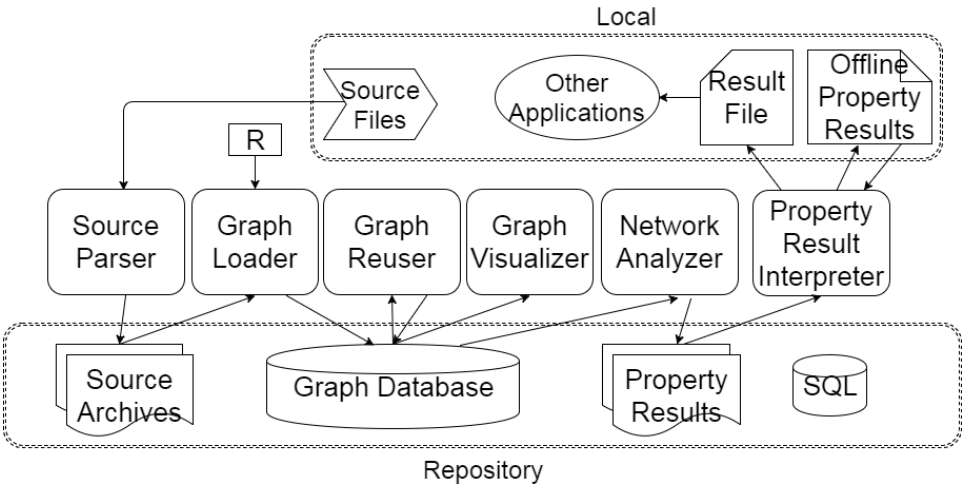


Figure 3.1: System Architecture

The repository holds all data the system uses. The three main datasets in the repository are source data, graph data and SNA property results. The source data are stored in source archive. The graphs are stored in the graph database. For different properties, the property results are in different forms, which are stored separately from the former two types of data. The repository manages the three large datasets in scalable clusters. The repository also has an SQL-based database to keep small datasets such as graph generating specification and calculation history. The system consists of six key components, which are source parser, graph loader, graph reuser, network analyzer, property result interpreter and graph visualizer.

The *source parser* is responsible for receiving users' data submission, parsing and saving the data to the source archive. The *graph loader* is responsible for reading data from source archive and generating SNA graphs in the graph database according to the user's specification. During the graph creation, the *graph loader* can call R to calculate weights of edges. The *graph loader* follows different graph generating processes for different data formats. The *graph reuser* is responsible for recognizing existing graphs for reuse, saving virtual graphs and generating virtual graphs. It checks graphs in the graph database and recognizes existing graphs which can be

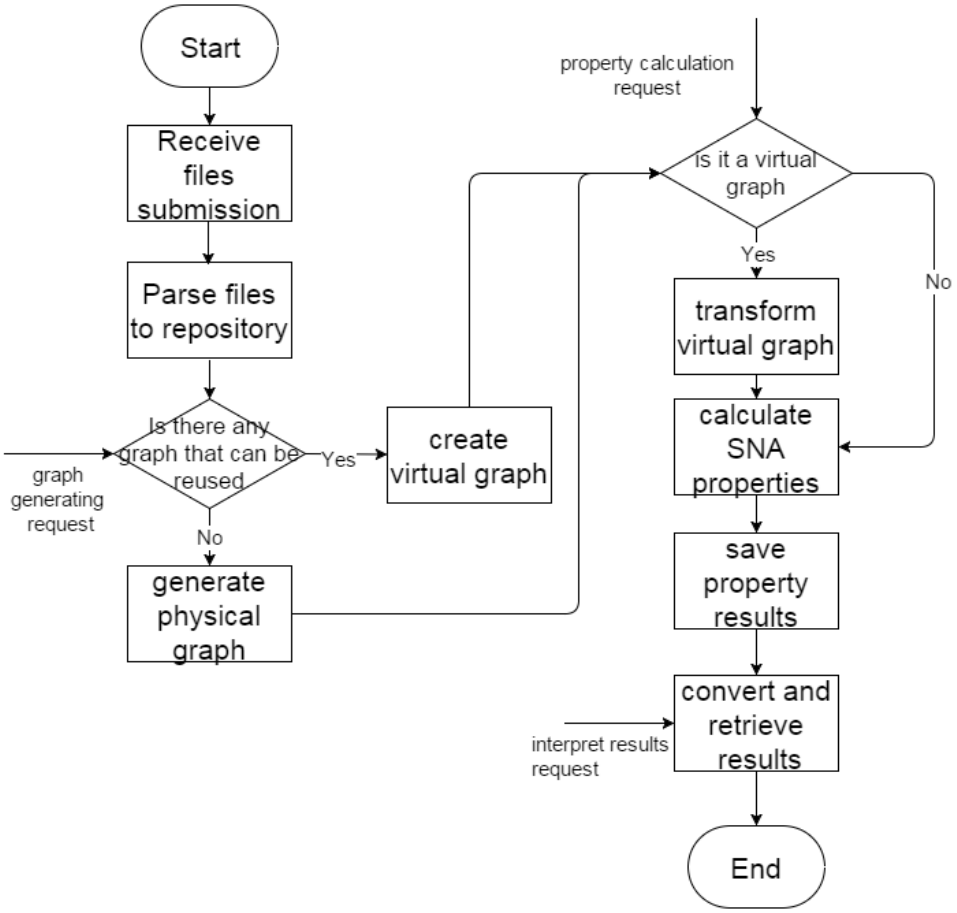


Figure 3.2: System Progress

used to get new graphs. When generating virtual graphs, it reads original graphs and rewrites new graphs back in the graph database. The *network analyzer* is responsible for offering property list, calculating SNA properties on graphs, saving property results and showing results to users. The *property result interpreter* is responsible for converting results in a number of formats and providing accessible results to users. The results can be retrieved from the repository and from files. The *graph visualizer* is responsible for visually displaying graphs in the graph database to users.

Figure 3.2 shows the workflow of the system. Next, we elaborate how the components interact with each other in the workflow. First, the *source parser* receives a group of source data files submitted by a user and parses the files into the source archive in the repository. Then the user submits the graph generating specification and

requests to generate graphs. The *graph reuser* component first checks if there is any created graph which can be reused to get the new graph. If not, the *graph loader* will read data from the source archive and generate a new graph in the graph database. If yes, the *graph reuser* will create a virtual graph by storing the virtual data in the SQL database. When the user requests to calculate SNA properties on the graph, the *network analyzer* first checks if the graph is a virtual graph. If yes, the *graph reuser* will make the virtual graph physical by transforming its associated graph in the graph database at first, then use the new created graph for property searching. If not, the *network analyzer* will calculate properties and saves the results in the repository. When the user asks for the results, the *property result interpreter* converts and retrieves the property results.

Chapter 4

System Implementation Framework

4.1 Data Collection

4.1.1 General Process

The source files are plain text in user's local machine. Data in the repository may be in a different file system, so users should submit the source files first.

Figure 4.1 shows the process of the dataset collection. It is managed by the source parser component.

The source parser receives a user data submission request. The request includes configuration such as source path, data format and a new archive name. The parser is configured and initiated. It creates a new archive path according to the given archive name, as all the data in the same submission will be stored under this archive path.

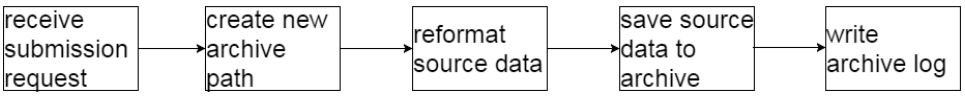


Figure 4.1: Data Collection Process

The parser reads the source files in the user's source path and reformats the data in order to support effective graph generation and search. For different data formats, the parser uses different data reformatting processes. The parser has a formatter for each data format to conduct the process.

The parser connects to the repository and saves the reformatted data to the source archive. After the source data are all stored in the repository, the parser writes the submission information to the archive log, which records the new archive path, data format and created time.

4.1.2 Data Format

The system supports reading data in two formats: time sequence format and transaction list format. Next, we elaborate the formats and how the source parser processes data in each format.

1. Time Sequence Format

A time sequence file structure is like Table 4.1. It presents a data unit or an actor. Each line is a key-value pair. The first line is the identifier of the data unit, followed by other attributes of the data unit if there is any. Then the time information is listed in the following lines, which contains a start time, an end time and time interval. Start time is the time at which the time series begins. End time is the time when the time series ends. Time interval represents how frequently a value is recorded, which can be daily, hours, minutes or seconds. The rest of the lines are time series values.

This format fits data whose values involve time. In the previously mentioned graph models, stock data fit in this format, as each stock has stock price which fluctuates over tims. In the Internet of Things (IoT), a set of time series data is collected every day from all types of sensors such as barometers and thermometers. It also has time series data and fits well in this format.

Table 4.1: Time Sequence Format

Identifier Attributes(Opt) Start time:20130103 End Time:20150106 Time interval:day 20130103:3.4 20130104:2.8 ....

A network is built from a group of files in time sequence format. When the parser transfers the data into the archive, it first creates a data unit table. Then it extracts the data units' attributes and time information and puts in the data unit table, as shown in Table 4.2. In the table, each row is a data unit. The row key is the identifier. In each row, the start time and end time are indexed. The parser also creates a time series table and puts time series data into the time series table, as shown in Table 4.3. Each row in the table stores a time-value pair. The time and units identifier are indexed.

Table 4.2: Data Unit Table

Unit	Start time	End Time	Time Interval	Attribute1	Attribute2
id	20130103	20150106	day		

Table 4.3: Time Series Table

Unit	Time	Value
id	20130103	3.4

2. Transaction List Format

Source files in transaction list format contain two text files, an actor file and a transaction file, as shown in Table 4.4. In the actor file, each row represents an actor. The row key is the identifier of the actor, such as id or name. The attributes

of the actor are key-value pairs in which there must be an attribute called type. The transaction file records a list of transactions between actors. Each row represents a transaction. The first and second columns are the identifiers of the two actors, such as email exchanges or coproduction times. The third column is the times or volume of their transactions following by the label name or other attributes of this transaction. This format is similar to the edge list with edge values. Data in many business data models can be categorized in the transaction list format. Information flows and exchanges, coproduction, agreements, directorships, transactions and all kinds of linked events data can be represented in the transaction list format.

Table 4.4: Transction List Format

(a) Actor File

Id:xx, type:xx, name:xx,.....
.....

(b) Transaction File

Actor1, Actor2, Time/Volume, Label(Opt).....
.....

When transferring data in the transaction list format, the parser creates an actor table and a transaction table in the archive. The data structures in the two tables are similar to the source files. In many cases, data analysts often use nodes' types to build and analyze graph models. Therefore, the system indexes each actor's type in the actor table.

4.2 Graph Generation

Graph generation is creating graphs in graph database from the source archive. Figure 4.2 shows the relationship between the components and other parts of the system during graph generation. The graph generating specification is given by the user which tells the system how to generate a graph. After the user submits a

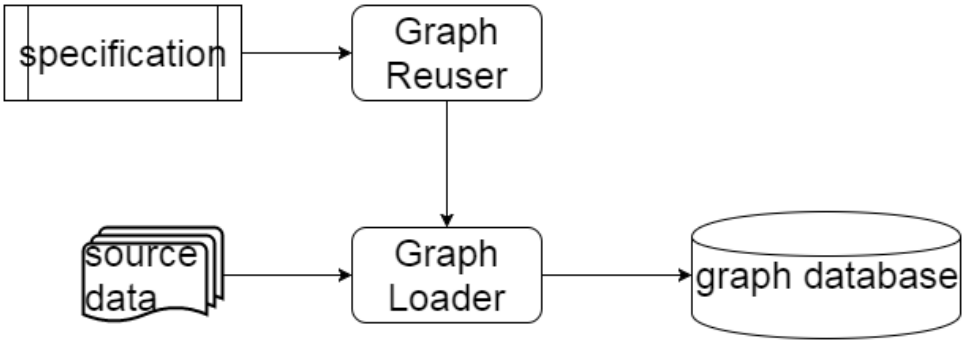


Figure 4.2: Graph Generation Context

specification, the graph reuser component will first check if the new graph can be transformed by any existing graph in the graph database. If not, the graph loader creates a new graph in the graph database by reading data from the source archive. For each data format, the graph loader predefines an automatic process to extract, transform and load graph data. Taking advantage of the data storage in clusters, the system can run MapReduce jobs to build large graphs.

In the previous part, we defined two data formats: time sequence format and transaction list format. Each format has its corresponding method to build graphs, so that the system can automate the graph generating process. Moreover, the system should always derive one unique graph based on one specification. Next, we give the specification details and elaborate the graph generating approach for each data format.

4.2.1 Graph Generation Specification

The common attributes of the specification give some general terms such as graph name, archive path input format, as shown in Table 4.5. For the time sequence format, the specification tells the start time, end time, and R function for building edges, as shown in Table 4.6. For the transaction list format, the graph is built by selecting different types of nodes and edges, so the specification tells what types of nodes and edges to create the graph, as shown in Table 4.7.

Table 4.5: Common Attributes in Graph Generation Specification

Attribute Name	Description
graphName	the name of the graph which should be unique in the system
archivePath	the directory name of the source archive, which should be created in data collection
graphDirection	whether the edges are directed or undirected. The value is true or false.
graphWeight	whether the edges have weights. The value is true or false.
inputFormat	the input format of the source, which is time seuqence or transaction list

Table 4.6: Attributes for Time Sequence Specification

Attribute Name	Description
startTime	the start time of the time sequence
endTime	the end time of the time sequence
Rfunction	the name of the R function
Rthreshold	the threshold for filtering the R results
weightMethod	the function of calculating weights based on R result values

Table 4.7: Attributes for Transaction Specification

Attribute Name	Description
nodeTypeSet	the set of the types for filtering nodes. It is optional. If not specified, all types of nodes are selected
edgeLabelSet	the set of labels for filtering edges. It is optional. If not specified, all types of edges are selected.
transactionThrehold	the threshold for filtering the trasaction times

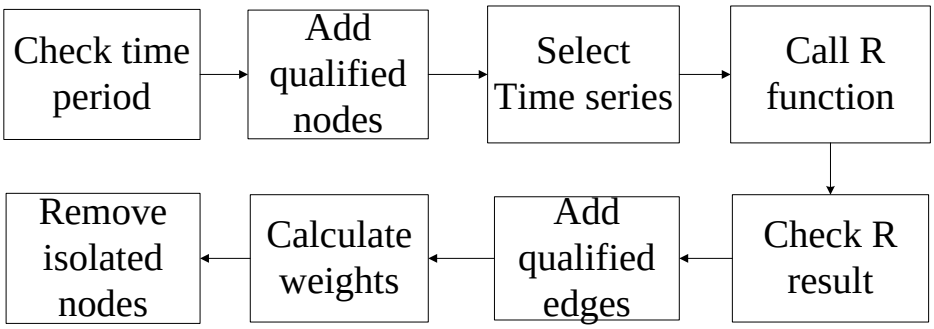


Figure 4.3: Building Graph for Time Sequence Format

4.2.2 Graph Generating Process

As shown in Figure 4.3, the steps of building graphs for the time sequence format are checking time period, adding qualified nodes, selecting time series, calling R function, checking result, adding qualified edges, calculating weights and removing isolated nodes. First the user selects a time period and an R function. The system considers each data unit as a node. The system first checks the start time, end time of each node in the data unit table. If the selected time period is within the time range of the node, it means that the node contains a valid series of data for the time period. Then the node will be added in the graph database.

For each node, the system selects the time series within the time period in the time series table. For each pair of nodes, the system calls an R function such as Pearson correlation or Kendall covariances, which inputs two sequences and returns a single value.

Based on the R function's returned value, the user sets up a threshold to decide if there is an edge between two nodes. If the value passes the threshold, the user tells how to calculate weights based on the value by setting a weight method such as log or absolute. Note that if the R function returns the same value after switching the order of the two input sequences, only undirected edge can be created. If it is not the

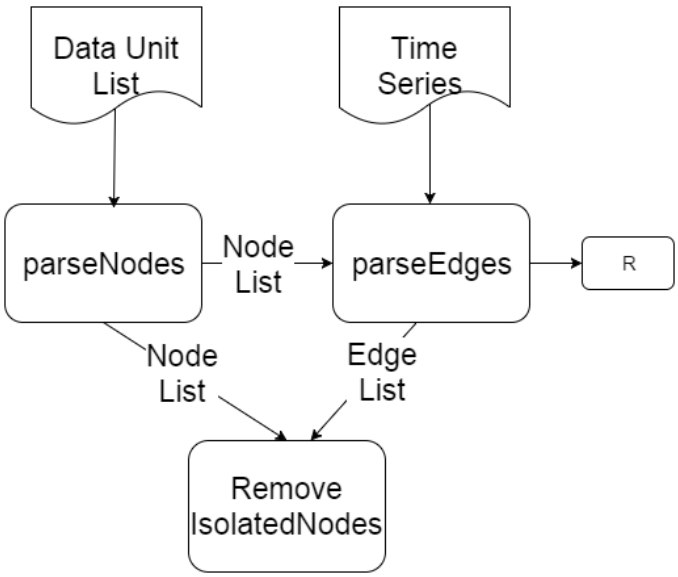


Figure 4.4: MapReduce Jobs

same, the system only creates directed edge from first node to second node. After the edges are created and the weights are assigned, the graph may have isolated nodes which have no edges connected to the other nodes because of the thresholds. Finally the system removes those isolated nodes from the graph. By selecting different time periods, users can compare networks' changes over times.

In the graph loader, we design three MapReduce jobs to complete the process, as shown in Figure 4.4. The first job is responsible for checking time period and selecting nodes from the data unit table. Its input is the data unit table, and the output is a list of qualified nodes. The second job selects two time series data for each pair of nodes, calls R, checks results and adds edges with weights. Its input is the node list produced by the first job and its output is a list of edges. The third job combines the outcomes of the two other jobs as input. It checks if there are some isolated nodes and removes those nodes. Its output is the final graph.

The generating process for the transaction list format can be simply represented as selecting nodes from the actor table and selecting edges from the transaction table. Users can select the types of nodes. The graph loader contains a node filter to

select and add the qualified nodes in the selected types. The user can also set up a threshold for the transaction times/volume which can be transformed to weights of edges indicating how well the actors associate with each other. The system reads line by line from the transaction table and creates edges whose weights are within the threshold. Finally, the selected nodes and edges form the outcome graph.

Similar to the time sequence format, the graph loader also has three MapReduce jobs for the transaction list format. The first job selects qualified nodes and outputs a node list. The second job outputs an edge list. The third job combines the node list and edge list results and outputs the final graph. It will remove edges whose start node or end node is not in the node list. It will also remove isolated nodes. As the actor table and transaction table are already similar to the forms of node list and edge list, building graph is easier for the transaction list format.

4.3 Graph Reuse

In this section we introduce our methodology of reusing generated graphs for SNA property searches. First we define some terms which are used in the process. Then we give the general graph reusing process that our system follows. Finally, based on each data format, we explain how our system decides situations by the graph generation specifications and how to derive new graphs.

4.3.1 Related Terms

We say that a physical graph is a graph stored in graph database, while a virtual graph is a logical graph but does not actually exist in the graph database. A virtual graph contains instructions by which we can build an actual graph by transforming a physical graph. A virtual graph is made into a physical graph for SNA property calculation.

As a new virtual graph is transformed from a physical graph, we classify their similar relationships based on three conditions. The first condition is that the new graph is a subgraph of the original graph, which means the new graph is a subset of vertices and a subset of edges of the original graph. The second condition is the new graph is a modified version of the original graph. It means nodes, edges or weights of the original graph have been changed. The third condition is that the new graph is a supergraph or an extended graph of the original graph, which means the original graph is a subgraph of the new graph.

The system decides how to transform graphs from one to another according to their relationships. The system checks if there is an existing graph in the repository which is a subgraph, a modified graph or an extended graph of the new graph by comparing their graph generation specifications. If it is a subgraph, the system will select a subset of nodes and edges from the existing graph. If it is a modified graph, it means we can replace one node or a subset of nodes, or edges or their weights. A similar term is graph rewriting which has a number of approaches and rules for graph transformation. But here we do not want our system to be too complicated, so we only consider changing weights in our system. If it is an extended graph, the system will add more nodes or edges to the original graph. Based on the above, if we understand the relationships between two graphs, we know how to transform one graph to another.

4.3.2 General Process

The graph reuse process is divided into three stages: checking existing graphs and deciding the relationship, adding the virtual graph and generating the virtual graph. Inside the graph reuse component, we design three subcomponents to complete the tasks: the reuse checker, the virtual graph saver and the virtual graph generator. Figure 4.5 shows the structure of the graph reuser component and how the component

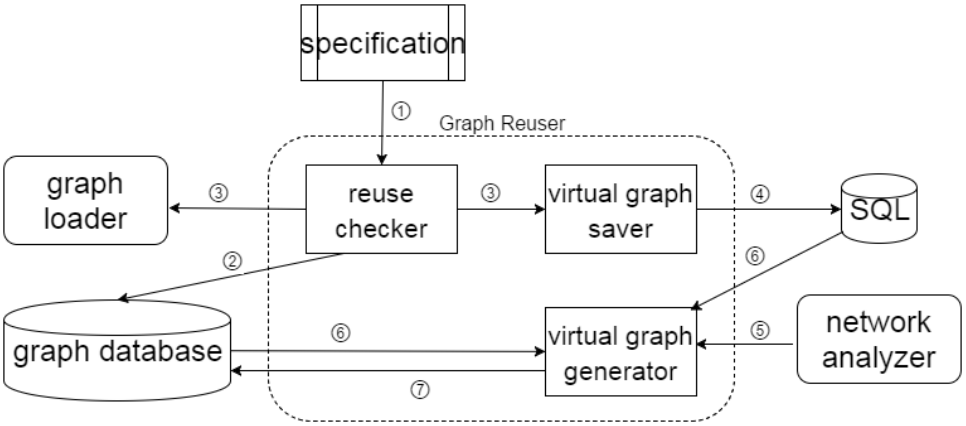


Figure 4.5: Graph Reuse Context

interacts with other parts of the system. The direction in the figure indicates the request or command from one to another. The order of the tasks are shown as the numbers in the direction arrows.

We elaborate the detail steps as follows:

1. When the user gives a new graph generation specification, it first goes to the reuser checker.

2. The reuse checker searches the specifications of the existing graphs in the graph database to find whether there is an existing graph of which the new graph is a subgraph, a modified graph or an extended graph.

3. If not, the reuse checker will forward the graph generating request to the graph loader. If yes, it means we can reuse the existing graph to get the new graph. Then reuse checker forwards the request to the virtual graph saver.

4. Then the virtual graph saver adds a virtual graph in SQL database by saving transforming information.

5. When the user wants to calculate SNA properties, the network analyzer first checks if the target graph is a virtual graph. If yes, it forwards the request to the virtual graph generator to get the physical graph.

6. The virtual graph generator reads the associated exiting graph from the graph database and the transforming information from the SQL database, and mutates the

original created graph to a new graph.

7. The virtual graph saver saves the new graph in graph database which will be used for property searching.

4.3.3 Virtual Graph Storage and Transformation

As we mentioned earlier, for graph reuse, the system should solve two key issues: how to decide graphs' relationship by their graph generation specifications and how to transforms graphs based on the virtual graph information. For each data format, the system defines an approach for determining graph relationships by comparing specifications. Then the system stores virtual graphs as reusing graph instruction in its repository. We define the structure of an instruction like this:

$$\{N, O, RELATION, \{K_1, V_1\}, \{K_2, V_2\}...\} \tag{4.1}$$

where N is the new graph. O is the original graph, $RELATION$ is the relationship. K is the changing key in the specification. V is the replaced value for the new graph. We may have more than one $\{K, V\}$, when we change both nodes and edges for example.

Next, based on each data format, we explain details of how to transform graphs for subgraphs, modifed graphs and extended graphs.

1. Time Sequence Format

When generating graph from data in the time sequence format, the user must specify source path, time period, R function, R result threshold and weighted method. For the same data source, if R function changes, it may indicate a different graph model. Also if the selected time period changes, it may also lead to changes of nodes and edges. Trying to find tradeoffs of space saving and time cost, we do not consider these changes to reuse graphs. Thus, we assume, when source path, time period, R function

are all the same, but R result threshold and weighted method are different, what is graphs' relationship and how we can transform graphs. There is one process in graph generation that we did not mention before. When the graph loader calculates R results, before applying the R threshold, it will store all the R results in the repository. Each R result list corresponds to a certain R function and time period, which will be used in the graph reuse process, so that the graph reuser does not need to call R when mutating graphs every time. We will explain more details below.

Subgraph: In the time sequence format, if the interval of R threshold of the new graph is within the R threshold of original graph, the new graph is a subgraph of the original graph before applying weight method. Thus the system stores the virtual graph's instruction like this: $\{V, O, subgraph, \{R_threshold, new_threshold\}\}$, where V means it is a virtual graph. O means the original graph, the relationship type is subgraph. The changing key is R threshold.

When generating the virtual graph, the system selects edges whose corresponding R results are within the new R threshold from the original graph. However, the weights of the edges are unlikely to be the same values as the R results, as a weighted method may change the result. Specifically, the system will use the original graph's R result list to select edges, and apply the weighted method to the qualified edges (if there is).

Modified graph: If the user specifies a different weighted function, while R result threshold remains the same, it indicates the weights in the graph are modified. Then the new graph can be counted as a modified graph. Thus we store an instruction in the repository like this: $\{V, O, modifiedgraph, \{weight_method, new_method\}\}$. The system applies the new weighted method on R results list of the original graph to change its weights and derive a new virtual graph.

Extended graph: If the interval of R threshold of the original graph is within the R threshold of the new graph, the new graph is a supergraph of the original graph before

applying weight method. Thus the system stores the virtual graph's instruction like this: $\{V, O, supergraph, \{R_threshold,\ complementary_threshold\}\}$, where V means it is a virtual graph. O means the original graph, the relationship type is supergraph. The changing key is R threshold. The value is complementary set of the threshold. It is the interval where values are in the new graph, but not in the original graph.

When generating the virtual graph, the system scans the original graph's R result list, and select edges whose R results are within the new graph's threshold but not included in the original graph's. The system adds the selected edges to the original graph. When calculating weight, the system still applies the same weighted method. Note that when adding edges, the system also adds their associated nodes in the graph.

2. Transaction list format

Subgraph: In the transaction list format, there are two situations that we consider for the subgraph. The first situation is that the selected types of the nodes in the new graph are a subset of the original graph. The system stores the virtual graph's instruction like this: $\{V, O, subgraph, \{NodeTypeSet, new_nodeTypeSet\}\}$, where the changing key is the node' types and the value is the new type set. When generating the virtual graph, the system selects nodes whose types are contained in the new types from the original graph. The second situation is the interval of transaction threshold of the new graph is within the transaction threshold of the original graph. As the transaction times are already listed in table like R result list, it is similar to the R threshold in the time sequence format.

Modified graph: As we do not have weighted method option for data in the transaction list format, there is no situation for modified graph.

Supergraph: In the transaction list format, there are two situations for the supergraph. The first situation is that the selected types of the nodes in the new graph

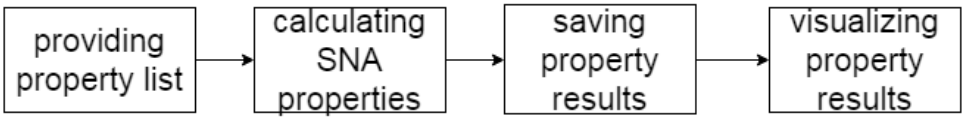


Figure 4.6: Property Searching Process

are a superset of the original graph. The system stores the virtual graph's instruction like this: $\{V, O, supergraph, \{NodeTypeSet,\ nodeTypeComplementarySet\}\}$, where the changing key is the nodes' types and the value is the complementary set of the type. When generating the virtual graph, the system adds nodes whose types are in the new graph's node type set but not in the original graphs. The second situation is the new transaction threshold has a wider range than the original transaction threshold. This situation is similar to the supergraph in the time sequence format.

We summarize the graph reuse situations in Table 4.8, 4.9.

4.4 SNA Property Searching

As shown in Figure 4.6, the SNA property searching follows a pipeline process: providing property list, calculating SNA property results, saving property results and visualizing property results. First the network analyzer provides a list of built-in SNA properties from which the user can choose the described properties to search on the graph. When doing property searching, the network analyzer checks if the graph is a virtual graph. If yes, it will call the graph reuser component to build the virtual graph in the graph database first. Then the analyzer reads graph data from the graph database and calculates properties based on the user selection. When the calculation is done, the analyzer saves the property results to the repository. The property result interpreter components can present the results to users. Next, we describe how the system works in each step.

Table 4.8: Graph Reuse for Time Sequence Format

Relationship	Situation	Instruction expression	Transfroming Process
Subgraph	Smaller range of R threshold	$\{V, O, subgraph, \{R_threshold, new_threshold\}\}$	1. select edges within new R threshold from R result list. 2. apply the weighted method
Modified graph	New weighted method	$\{V, O, modifiedgraph, \{weight_method, new_method\}\}$	Apply the new weighted method
Supergraph	Wider range of R threshold	$\{V, O, supergraph, \{R_threshold, complementary_threshold\}\}$	1. add edges in the new R threshold. 2. apply weighted method

Table 4.9: Graph Reuse for Transaction List Format

Relationship	Situation	Instruction expression	Transfroming Process
Subgraph	Subset types of nodes	$\{V, O, subgraph, \{NodeTypeSet, new_nodeTypeSet\}\}$	selects nodes whose types are in the new types
	Smaller range of transaction threshold	$\{V, O, subgraph, \{T_threshold, new_threshold\}\}$	select edges within new transaction threshold
Modified graph	N/A		
Supergraph	Superset types of nodes	$\{V, O, supergraph, \{NodeTypeSet, nodeType\ ComplementarySet\}\}$	add nodes with the new types
	Wider range of transaction threshold	$\{V, O, supergraph, \{T_threshold, complementary_threshold\}\}$	add edges in the new transaction threshold

4.4.1 Property List

Some SNA properties can be used to calculate on both directed and undirected graphs. However, some can only be used in one type of graphs. For example, the page rank is only used on directed graphs, while eigenvector centrality is only for undirected graphs. Therefore, the system provide different property lists for a directed and an undirected graph. When showing the property list, the user first chooses a graph, and the system returns proper SNA properties according to the graph's type.

4.4.2 Property Calculation

The system uses Google's Pregel model for SNA property searching. The Pregel supports implementing arbitrary graph algorithms over arbitrary graph representations on large-scale graphs for general purpose [19]. The property searching algorithms implemented in Pregel are run as parallel graph processing programs, which is required by the system's requirement. We use Pregel rather than MapReduce, as Pregel's APIs are more suitable for natural graph processing and more efficient. Although MapReduce can also run parallel graph algorithms by passing the whole graph in mapreduce chain, it requires more communication and serialization work.

In contrast, the computation in Pregel is composed by several supersteps. In each superstep, vertices receive messages from their neighbors, update their instinct values and send new messages through their outgoing edges. The vertex is active or not by voting to halt. It will stop working if it does not receive messages from its neighbors. The computation will stop until there is no message sending among the graph. This computation framework is suitable for calculating SNA properties like degree centrality, page rank, and single-source shortest path, as messages iteratively are sent, received and updated through vertices, and each vertex acts as a working unit computing simultaneously.

The network analyzer defines a property program for each property which overrides Pregel's *Compute*() method. When executing the property calculation, the Pregel will copy the property program to clusters, partition graph and assign to each working machine, and instruct workers to execute *Compute*() method in each superstep. Note that, a property program can execute different algorithms in the *Compute*() method for a weighted and unweighted graph.

4.4.3 Property Result

The output formats of property calculation are various. The result can be a single value such as the average degree. It can also be a set of vertices from a large graph such as community detection. Therefore, we classify property result as network level property result and node level property result. The network level property result is categorized into simple network result and network statistics result. A simple network result is presented as a single map which is a key-value pair. The network statistics results are aggregated statistics which may be large unstructured datasets. The node level property results are a set of values per-vertex which may also be large datasets. Table 4.10 shows the classification of property results and the property examples.

Table 4.10: Classification of Property Result

Property result's type		Example
Network level property results	Simple network results	Diameter, average degree, shortest path
	Network statistics results	Degree distribution, path distribution, clustering coefficient distribution, community structure
Node level property results		Degree centrality, betweenness centrality, closeness centrality, eigenvector centrality, page rank, clustering coefficient

Each type of property results are stored in a different form and managed by different

storage. In the repository, the simple network results are stored as maps logically, while the node level property results and network statistics results are stored as lists. The simple network results are kept in a local file system, while the node level property results and network statistics results are managed by the distributed file system.

4.4.4 Viewing Property

The property result interpreter provides a calculated property list. The user chooses to view property results from the list. The interpreter shows property results by the user selection. We will explain more details of the interpreter in the next section.

4.5 Business Application Interpretation

Business application interpretation offers property results for users to integrate to their applications. The user inputs the property name and the format of the property results to the property result interpreter component. Then the interpreter retrieves the property results in the designated format. The interpreter offers two ways of accessing property results: online and offline. The online way gets results from the repository while the offline way gets results from local files.

4.5.1 Result File Hierarchy

In the repository, SNA property results are organized as a group of files in hierarchy, as shown in Figure 4.7. The top level file is a property overview file which lists all the properties of the graph. The property overview file contains simple network results listed as key-value pairs. In lower level, the network statistics results and node level property results are stored in separate files, which are referenced by the overview file. While searching the results, the interpreter begins looking at the property overview

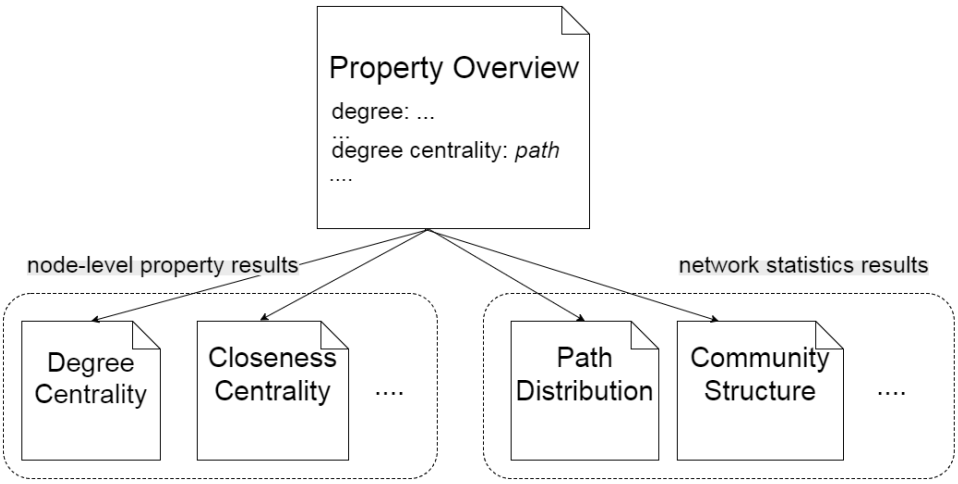


Figure 4.7: File Hierarchy

and traverses to other corresponding result files. While getting results offline, the local result files are also stored in this file hierarchy.

4.5.2 Interpretation Formats

We classify the common formats of property results in different property interpretation scenarios, as shown in Table 4.11. These formats are common to be output as files which are to be read by other applications as input or to view.

Table 4.11: Classification of Result Interpreting Formats

Interpreting Formats	Example Properties
graph	clustering, shortest path
matrix	centralities
list	centralities, degree distribution
map	diameter, average shortest path
JSON	clustering,centralities

To be interpreted to *graph format*, property results are written to the original graph or a new graph. The results may be formed by elements in the graph such as a path or clusters of vertices. The results can be written back to the graph for continuous semantic analysis. The *matrix format* is for node level property results. In the matrix format, the node level property results are grouped together for a vertex in

each line. It can be used to calculate the structure embeddedness of a vertex in the graph. The *list format* can be used to present all kinds of properties. The *map format* is a key-value pair, which is suitable for values of small dataset, such as simple property results. The *Json format* looks like a combination of lists and maps. There can be multiple JSON representations. Potentially, any property results can be presented by JSON.

The interpreter's work can be divided into two parts: converting results and returning results. The system provides two ways to interpret property results. The online way is converting and retrieving results from the repository. As mentioned in the last section, the network analyzer outputs the property results in the repository. The user sends the interpreting results request. The request contains the property and selected output format of the results. The system predefines a number of output formats the system can convert as mentioned above. The default format of the result is defined by the result's type. The simple property results are stored in map format. The node level property results and the statistics network results are stored as list. After the format conversion is done, the results are still in the repository, and then the interpreter returns the results according to their formats. If the results are generated in a file such as a list or a matrix, the interpreter retrieves the result file to a user's path. If the format is a graph, the interpreter returns the graph path in the graph database, and the user can read the properties using the graph database API. The offline interpretation allows users download all property results after the calculation and access the results in the local machine. The system supports limited interpreted formats as the formatting work is executed locally. As the results are already in local path, the system can return the results to users directly.

Chapter 5

Prototype System Implementation

A prototype system has been implemented on a Hadoop cluster. The prototype implementation implements the five major modules: data collection, graph generation, graph reuse, property searching and business application interpretation, as described in the implementation framework.

The implementation is based on the client-server architecture. As users are clients connecting to the system's server. The server is built on a Hadoop cluster as the server acts as the master of the cluster; it receives users' request and executes data processing on the cluster.

The repository combines distributed mode and local mode physical storage, as shown in Figure 5.1. For storing large data in various structures, its physical layer includes several databases. The distributed storage is mainly based on HDFS which is capable of managing vast quantities of data in clusters. Based on HDFS, we apply HBase which supports quick access and search at scale to store source archives. We use Titan as our graph database for handling distributed graphs. We use HDFS to save property results. We also set up HBase as Titan's storage backend. Therefore, to set up the repository on cluster physically, we first deploy HDFS as the base. Upon HDFS, we install HBase to manage source data. Based on HBase, we build Titan

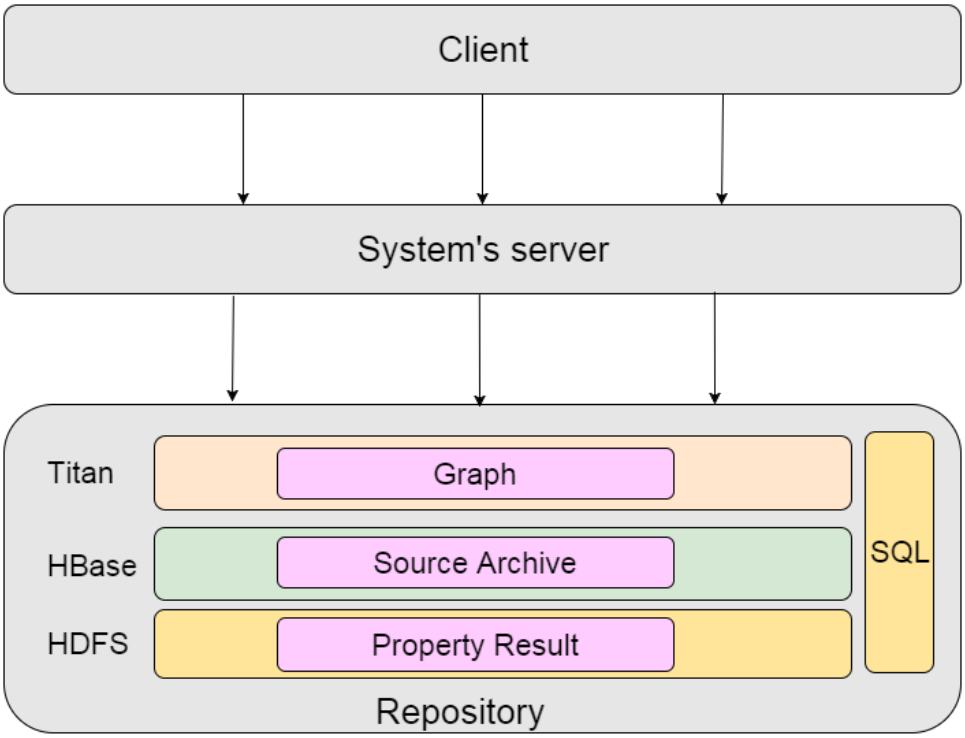


Figure 5.1: The Physical Structure of the Repository

for graph storage. The property results are mainly stored in HDFS. The HBase and Titan's APIs are integrated in the system to access data in the repository. On the local mode, we use an SQL database to keep all the graphs' configuration, virtual graph profile and other related information.

In the prototype system, as the transaction format is simpler, we mainly focus on building graphs on the time sequence format for datasets. For the property searching, we implement four SNA property calculations: degree centrality, closeness centrality, betweenness centrality and eigenvector centrality. For property interpretation, the prototype system supports graph, list and matrix formats. The prototype system builds on Java and offers Java APIs for users' application integration, as Hadoop, HBase and Titan all use Java as the native language. The versions of the software platform used in the prototype system are Apache Hadoop 2.6.0, Apache HBase 0.98 and Titan 0.5.4.

5.1 Data Collection

The system uses HBase to store source data. HBase is based on Google's BigTable model [46]. It uses table to store data; a table has rows and columns. Each row has a row key and a number of columns. Each column has a column family and a column qualifier. A column family groups a set of columns. Each of the columns in a column family is distinguished by the column qualifier. In an HBase table, each row has a fixed number of column families, but can have arbitrary column qualifiers. Thus, it is suitable to store a data unit which has arbitrary number of attributes. Rows in an HBase table are sorted in alphabetical order of row keys. Taking advantages of it, we design the row keys in our archive tables by how they will be queried to support efficiency. Next, we introduce the schema of the tables for time sequence format in the system.

As mentioned in the previous chapter, for data in the time sequence format, the system has two tables: data unit table and time series table. The data unit table stores the units' time information and related attributes. The time series table stores a series of values of each unit. In HBase, for each source data submission, the system creates a data unit table to keep all the data units' information and creates a time series table to keep all the time series values for all the units. Table 5.1 defines the schema of the data unit table. In the data unit table, the row key is the identifier of a data unit. The table defines two column families: *timeInfo* and *attribute*. The *timeInfo* column family has three qualifiers: *startTime*, *endTime* and *timeInterval*, which correspond to the start time, end time and time interval of the unit respectively. The *attributes* column family has an arbitrary number of column qualifiers. Each qualifier corresponds to an attribute of the unit which is contained in the source files. Table 5.2 defines the schema of the time series table. In the time series table, each row stores a time and a value of a unit. It only has one column family named *timeSeries*, and one qualifier in the column family named *value*. We

define each row to store a single time series value, not the entire time series values of a unit. HBase prefers a tall narrow table than a wide table for efficiency, as the entire time series of a unit can be very long. The row key combines the identifier and the time. When the system wants to select a time period for a unit, the time in the row key is indexed for quick searching, as the table is sorted by its row key. And the identifier before a time is to avoid hotspotting.

Table 5.1: Data Unit Table Schema

row key	timeInfo			attribute	
identifier	startTime	endTime	timeInterval	attribute1	attribute2

Table 5.2: Time Series Table Schema

row key	timeSeries
identifier#time	value

Figure 5.2 shows the class diagram for the implementation of the source parser component. The SourceSubmissionReceiver receives the user's source files submission with the source files path, a source archive name and the data format. It calls the SourceConsumerFactory to get an SourceFileConsumer according to the data format. The SourceFileConsumer is an interface which has two methods to be implemented. The process method first calls createArchive() to create new archive tables for this submission and copy the data into the tables. The TimeSequenceConsumer implements the sourceFileConsumer interface to process time sequence format. Users can also define a new class to implement this interface for a new data format. The TimeSequenceConsumer holds a TimeSequenceFormatParser which does the substantial work for parsing data unit information and time series data. The HBaseHandler connects directly to the HBase and creates records in the table. The TimeSequenceConsumer first creates a data unit table and a time series table in the HBase. Then for each file, the TimeSequenceConsumer iteratively extracts the data unit information and time series data, and stores in the data unit table and the time series

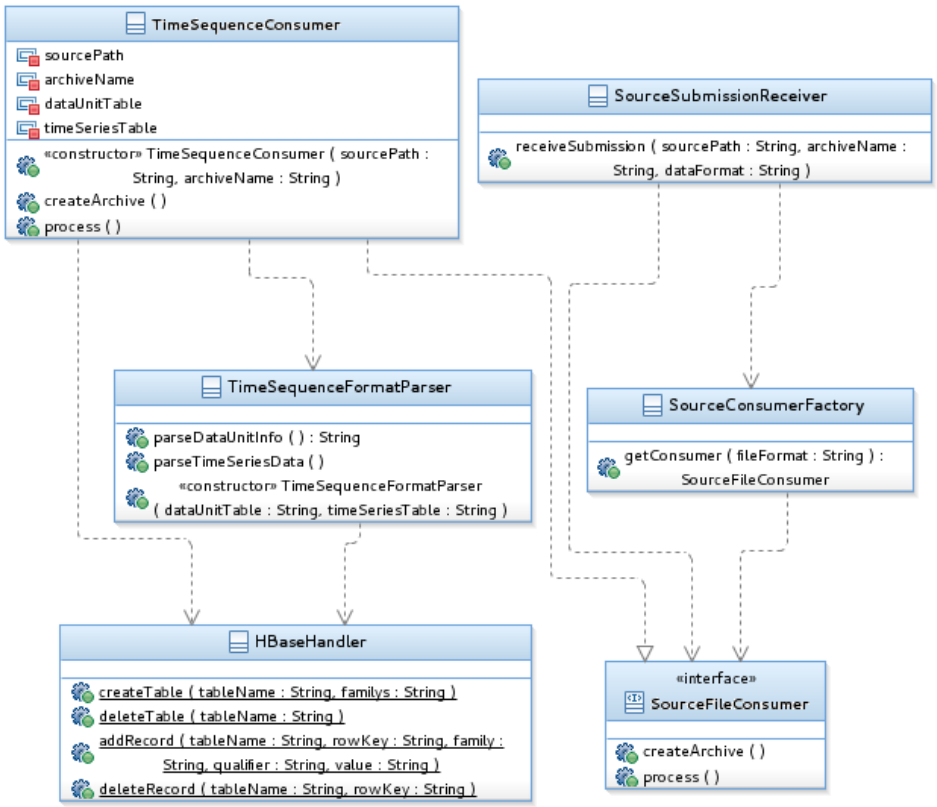


Figure 5.2: The Class Diagram of the Source Parser Component

table.

5.2 Graph Generation

Figure 5.3 shows the class diagram for general graph loader component. The GraphLoader interface is responsible for the general graph generating process. The BaseGraphLoader implements the interface which holds GraphLoaderProperties containing graph generating specification. The combineProperties() method which combines the graph generating specification with Titan's graph configuration, before the generating process starts. The saveGraphConfToRepo() method saves the graph generating configuration to the repository. We can create graph loader subclasses extending the BaseGraphLoader class for different data formats. A graph loader subclass defines an

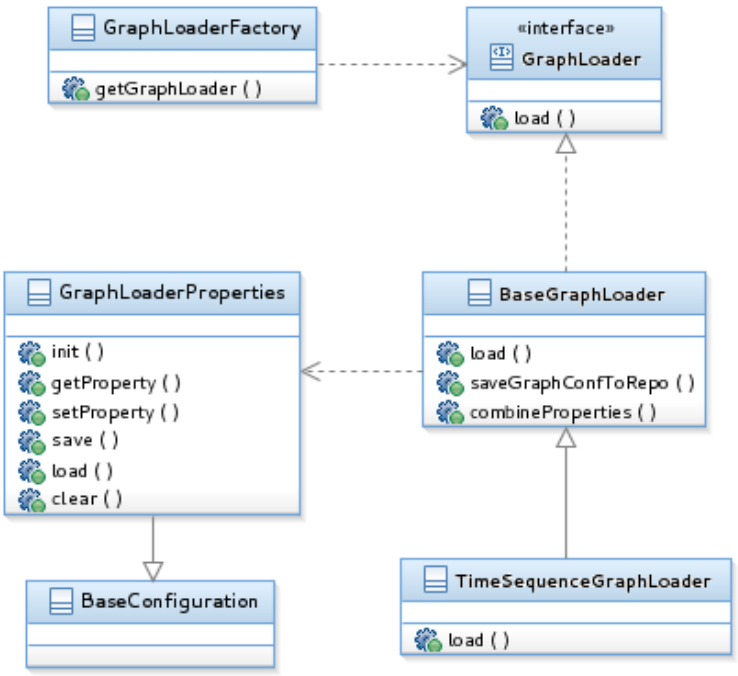


Figure 5.3: The Class Diagram of General Graph Generation

automatic process to ETL (extract, transform and load) the data from the archive to the graph database, and conducts the graph generation by running a series of MapReduce jobs.

Next, we elaborate how the TimeSequenceGraphLoader is implemented for building graphs from time sequence formated data.

As described in the previous chapter, the loader class has three jobs to complete the task: a parsing nodes job, a parsing edges job and a removing isolated nodes job. The first two jobs read nodes and edges information from HBase and create them in Titan. The Titan's 0.5.4 version contains Titan-Hadoop package which is a graph analytics engine running with built-in Mapreduce on Titan graphs. Users can choose and configure the IO format of Titan-Hadoop, such as GraphSON, script or Titan's HBase. As the backend storage of Titan is HBase, the output format is TitanHBaseOutputFormat. The input should be the HBase tables in the repository. However, Titan cannot read the tables directly. Although Titan-Hadoop has a script

IO format which users can read graph data by writing their own script, the method does not support reading HBase as in our case. It only supports reading a plain text. Therefore, to save the graph into Titan by using Mapreduce, there are two ways to do it. The first way is to write our own input format and mapreduce jobs that Titan-Hadoop can execute. The other way is to assemble a file in a certain format which Titan-Hadoop's input format API can read at first, then run Titan-Hadoop's built-in Mapreduce to load the graph. Here we use the second way to implement in the system. Although the first way may be an ideal way, writing and running our own MapReduce job inside Titan is complicated, and we decide to leave it as a future work.

The general method to load a graph for the time sequence format is to assemble a text file in adjacency list format and using Titan-Hadoop's script input format API to read it into Titan. Figure 5.4 shows the class diagram of the graph loader component for time sequence format.

The TimeSequenceGraphLoader takes four steps to load the graph. First, it reads data from the data unit table and outputs a node list. Second, it inputs the node list, calculates R results for each pair of nodes and outputs an adjacent list file. Third, it calls Titan-Hadoop traverse method to read the adjacency list file and builds the graph in Titan. Fourth, it uses Titan-Hadoop filter method to remove the nodes which have no neighbors.

The TimeSequenceGraphLoader has four jobs corresponding to the four steps: parseNodeJob, parseEdgeJob, ScriptInputJob and RemoveIsolatedNode. We explain the implementation details of each job below.

The ParseNodeJob uses Hadoop ToolRunner to run the job by implementing Hadoop's Tool interface. It contains a single map ParseNodeMapper to process data in HBase table. It sets up two HBase Scan filters to select nodes in the data unit table. The mapper extracts the attribute information in each row and writes them in the node

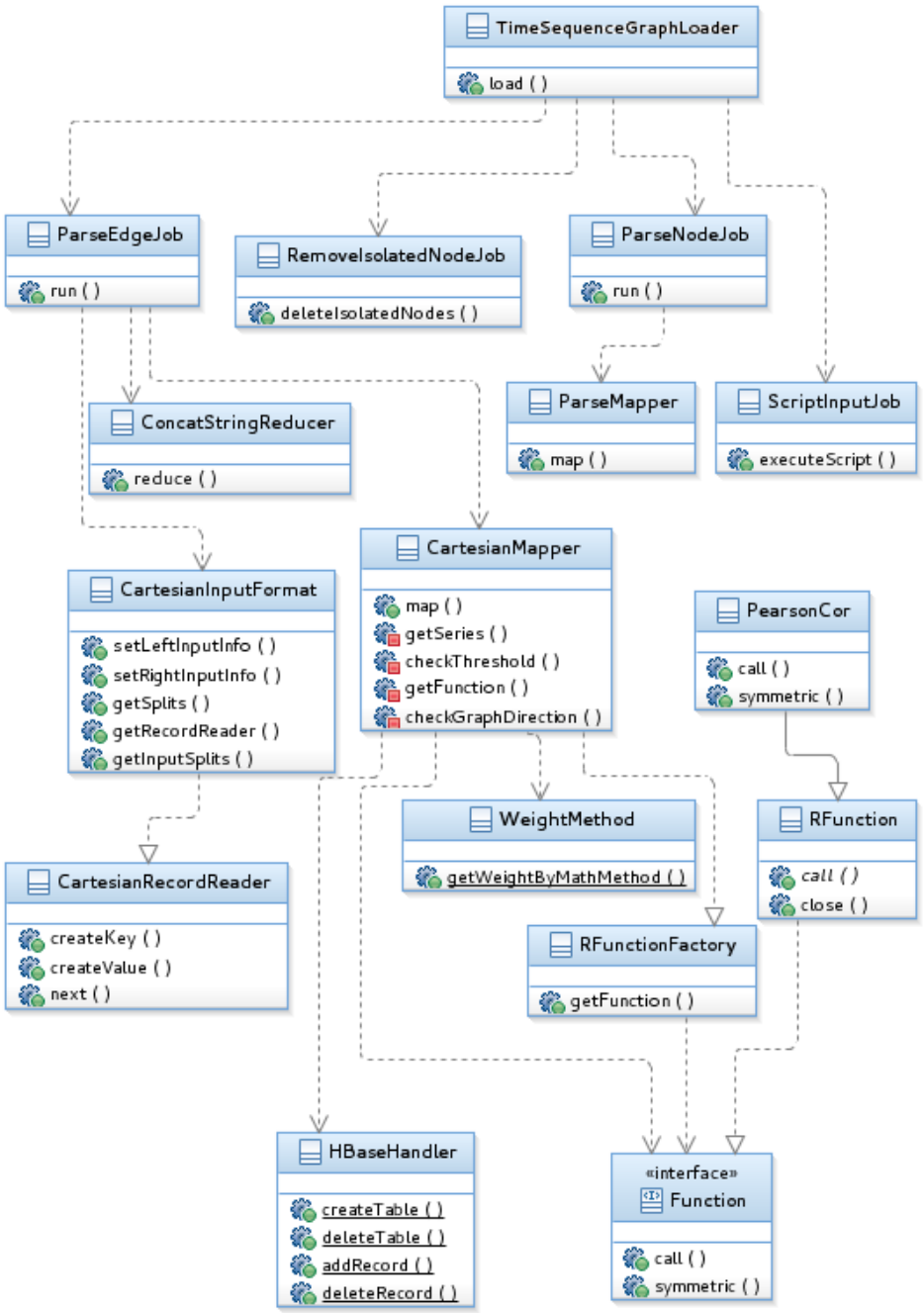


Figure 5.4: The Class Diagram of Graph Loader Component for Time Seuqence Format

list file. The file is a plain text with each line representing a node.

The ParseEdgeJob gets R results for each pair of nodes in the node list by reading time series data from the time series table. If an R result passes the R threshold, an edge will be added to the output file. The output file is an adjacency list in which each line starts with a source node following a list of target nodes with weights representing edges between the source node and target nodes.

As we need to calculate every pair of nodes, it fits well in the MapReduce Cartesian product design pattern. We use the Cartesian Product example in [47] which defines a CartesianInputFormat and a CartesianMapper. The CartesianInputFormat reads the node list splits and sets a left dataset and a right dataset with the same node list. The CartesianRecordReader performs the cross product by pairing the left dataset to each one of the right dataset. It produces a key-value pair which can be processed by the map. The key is one record from the left dataset and the value is one from the right dataset. One record represents one node in the node list. For each nodes pair, the CartesianMapper calls getSeries() to select time sequence data from the time series table and calls R function to calculate.

The Rfunction class implements Function interface. The Rfunction contains a REngine which is a singleton and is responsible for connecting R through JRI. Under RFunction, there are a number of subclasses as each subclass implements a specific function, such as pearson correlation or spearman correlation. In the diagram, the PearsonCor is an example subclass. The call() method returns results. The symmetric() method which returns true if the function is symmetric. If it is symmetric, the system will create two directed edges for both directions. Otherwise, a directed edge from the start node to the target node is created. The CartesianMapper calls checkThreshold() to check if the R results passing the threshold, and calls WeightMethod's weighted method to compute weights.

The mapper writes the output map with the start node identifier to be the key and

the target node with their weights to be the value. If the R result of the two nodes does not pass the R threshold, the output value is null indicating there is no edge between the nodes. During the CartesianMapper outputs to the reducer, the mapper also outputs an R result list to the repository through MapReduce MultipleOutputs which stores all the original R results between pairs of nodes. The R result list is to be used in the graph reuse process.

The ParseEdgeJob has a ConcatStringReduce to process the output of the CartesianMapper. It concatenates the string values with the same key, so the target nodes with the same start node are put in the same line when being added to form an adjacency list. In the adjacency list file, the row key is the start node following by the attributes of nodes and the target nodes.

The ScriptInputJob parses the adjacency list into Titan by using Titan-Hadoop's ScriptInputFormat. The system predefines a groovy script which deciphers each line for Titan vertex. It reads a predefined Titan HadoopGraph configuration which ties the adjacency list with the groovy script file, and executes the HadoopGraph built-in MapReduce jobs to bulk load graphs. The RemoveIsolatedNodeJob holds a HadoopGraph which inputs and outputs Titan's graph in HBase. It calls HadoopPipeline filter and drop method to deletes all the isolated nodes from the graph.

5.3 Graph Reuse

We divide the graph reuse process into two phases: virtual graph saving and virtual graph generation. We give the class diagram for each phase below.

Figure 5.5 shows the class diagram for virtual graph saving. When a graph generating request comes, the GraphReuseChecker acts as a filter before the graph generation. Based on the new graph generation specification, it searches all created graphs in the repository, checks their specifications and tries to find a matching one for reuse. If

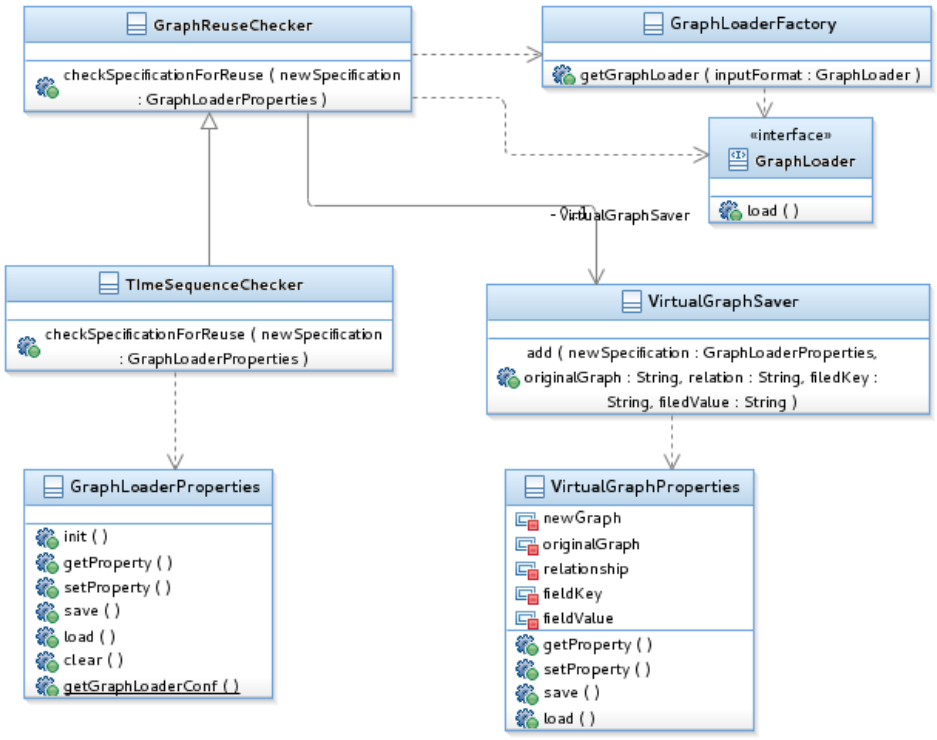


Figure 5.5: The Class Diagram of Virtual Graph Saving

there is no graph matching, it will call the GraphLoader's load() method to generate a new graph. If there is a reusable graph in the repository, the GraphReuseChecker calls the VirtualGraphSaver to save the virtual graph profile in the repository.

Each data format has a checker. In the class diagram, the TimeSequenceChecker which extends GraphReuseChecker checks graph generation specifications for the time sequence format. The checkSpecificationForReuse() method contains the logic to find graphs for reuse. It first checks if there is any created graph which uses the same time period and R function with the new graph. If there is one, it compares their weighted method and R threshold in three cases. If their weighted methods are the same but the new graph's R threshold's range is within the existing one, the GraphReusechecker calls the VirtualGraphSaver to add a new virtual graph as a subgraph of the existing graph. If their weighted methods are the same but the existing graph's R threshold's range is within the new graph's, the VirtualGraphSaver



Figure 5.6: The Class Diagram of Virtual Graph Generation

adds a new virtual graph as a supergraph of the existing graph. If their R thresholds are the same but the weighted methods are different, a new virtual graph is added as the modified graph of the exiting graph. When calling the add() method to add a virtual graph, it gives the parameters to the virtual graph profile including the new graph generation specification, its associated existing graph, their relationships, their differed keys and values.

Figure 5.6 shows the class diagram for virtual graph generation. In the second phase, before the SNAPropertyCalculator does property calculation, the system checks if the target graph is a virtual graph. If yes, it calls the VirtualGraphGenerator to convert a virtual graph into a physical graph in the graph database. The VirtualGraphGeneratorForTimeSequence generates a physical graph from a virtual graph for the time sequence format. It reads the virtual graph profile and calls the GraphTransformer to process and transform the graph. The GraphTransformer has three

methods corresponding to the three graph reuse cases. For the subgraph and supergraph cases, the mapper has output the R result list at its first running time. The getSubgraphByRThreshold method generates the virtual-to-physical graph by selecting a subgraph from the original graph with the new R threshold. The getModifiedGraphByWeight method generates the graph by altering the edges' weights with the new weight method. The getSuperGraphByThreshold method adds new edges whose weights are within the new R threshold but are not included in the original graph. In the original graph, the values of weights may not equal to the original R results as changed by the weight method, to avoid recalculates time series data, the GraphTransformer applies the new R threshold or the weight method to the R result list to derive the new graph.

Ideally, these graph transformations are run inside Titan by mutating the original graph using MapReduce. However, Titan-Hadoop does not provide an explicit subgraph function, nor does it well support in other graph mutations. Also in our case, the MapReduce job should also read the R result list when mutating graphs. Thus, our system uses GremlinPipe [48] to manipulate graph transformations in Titan, which is used for real-time traversals.

5.4 SNA Property Searching

In the property searching, we classify the implementation into two layers: the framework layer and the detailed layer. The framework layer provides a framework for choosing and running SNA properties for searching. The detailed layer does actual computations on SNA graphs.

Figure 5.7 shows the class diagram for property searching. The SNAPropertyListBuilder offers a list of properties users to choose for searching. It provides separate property lists for directed and undirected graphs. The SNAPropertyCalculator con-

ducts the property searching process. It first checks if the graph is a virtual graph before calculation begins. Then it calls calc() method to calculate the user selected properties one by one. It gets a SNAProperty from SNAPropertyFactory. The SNAProperty provides the interface for a SNA property calculation. Any property searching program run by the system implements the interface. The class diagram shows four SNA property classes for degree centrality, betweenness centrality, closeness centrality and eigenvector centrality. Each class contains different algorithms for a weighted graph and an unweighted graph.

Next, we explain the lower part of property searching including concrete property calculation, results saving and viewing.

For graph calculation, the system uses Apache Giraph [20] which is a graph processing framework implementing Pregel model in MapReduce. It runs Map-only job in cluster to perform supersteps, as each worker keeps its own splits in memory and processes them in parallel. As the system is already built on Hadoop cluster, we easily integrate Giraph in the system. We also write our own SNA property searching algorithms under Giraph framework. The algorithm class extends the BasicComputation class and override the compute() method. The compute() method contains the core algorithm to be executed on each vertex iteratively. In the system we use the original Giraph package. Although the Tinkerpop 3's Hadoop-Gremlin which is the advanced version of Titan-Hadoop provides running Giraph inside, it is not suitable for our system. There are two reasons. The first one is that Tinkerpop3's buit-in Giraph does not support analyzing edges with weights currently. The second is that the Tinkerpop3 only supports Titan 1.0 API, and it only can read external files in three formats: Graphson, Gyro and script IO. The Titan version that the system uses is 0.5.4 which is not compatible with the Tinkerpop3. Although we can still output a graphson file from our Titan graph database and ask Hadoop-Gremlin to do the calculation, its current built-in Giraph version is 1.0 which runs on Hadoop

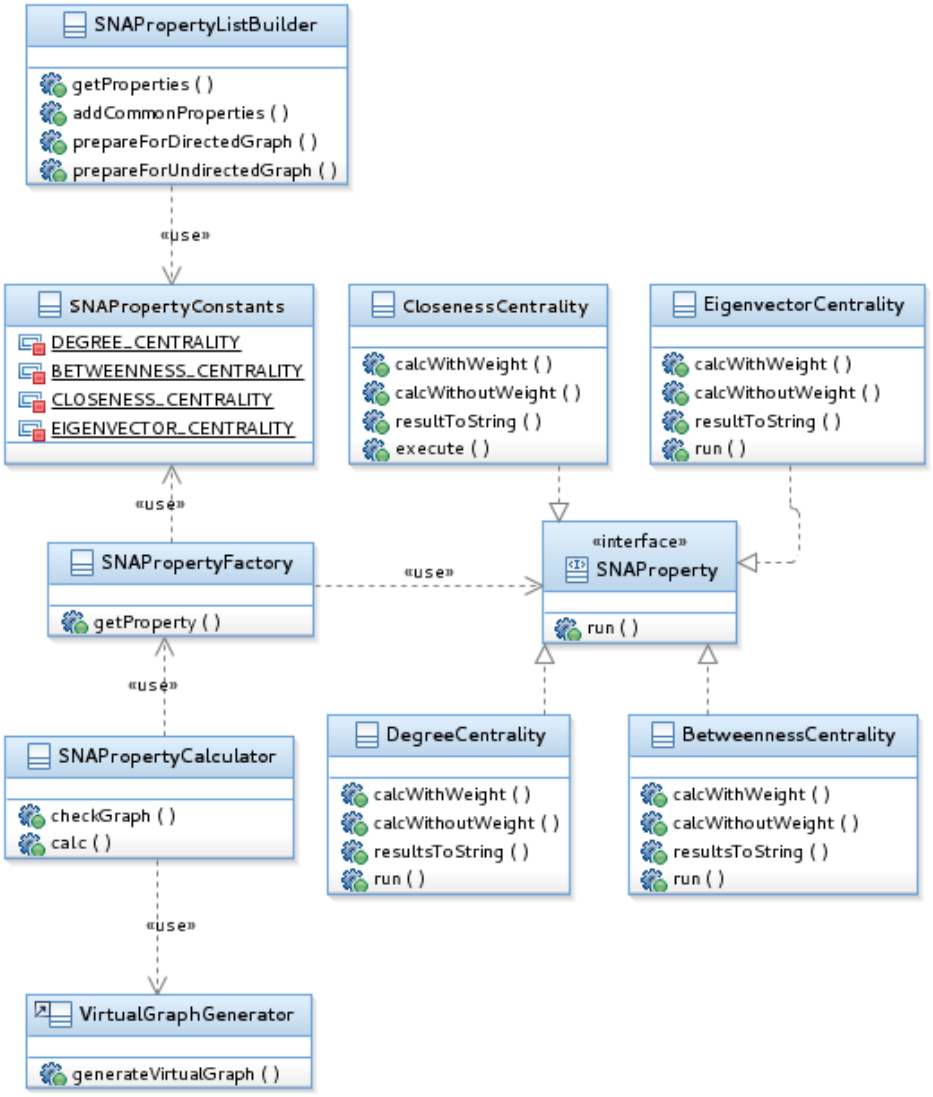


Figure 5.7: The Class Diagram of Property Searching

1 and not Hadoop 2. Therefore it is not better than using the Giraph directly.

Figure 5.8 also demonstrates the class relationship for calculating eigenvector centrality. The EigenvectorCentrality uses ToolRunner to run a GiraphJob by implementing the Tool interface. The system uses Giraph 1.1.0 which supports Hadoop Yarn. It configures a Giraph job by setting the EigenvectorCentralityComputation class as the Giraph job's ComputationClass. The EigenvectorCentralityComputation's compute() method will be executed on each vertex for each superstep. The Giraph job's

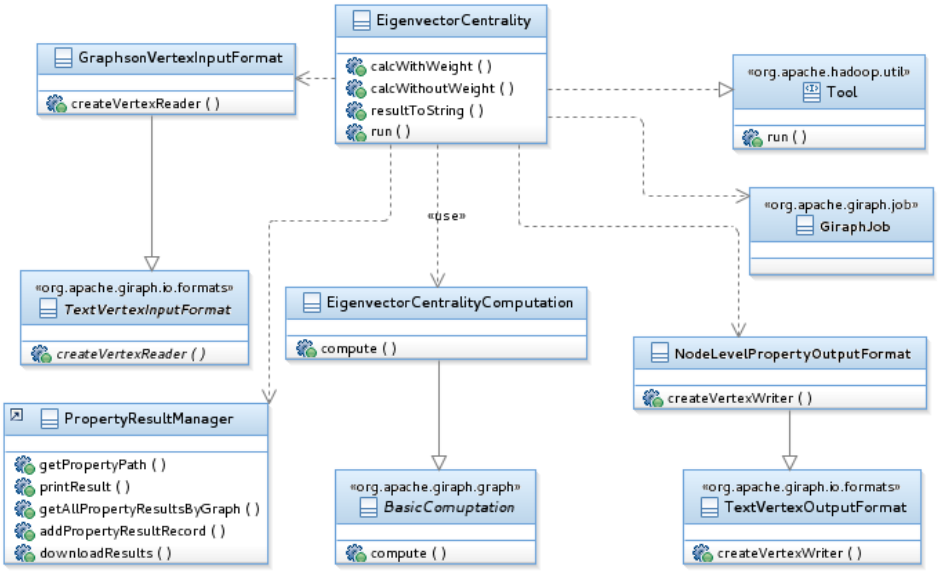


Figure 5.8: The Class Diagram of Eigenvector Centrality Calculation

input format is GraphsonVertexInputFormat and its output format is NodeLevelPropertyOutputFormat. Before the property computation starts, the system first outputs the graph data from Titan as a graphson format so the InputFormat can read. The output format is a property result list. In an eigenvalue list, each line contains a vertex and its eigenvector centrality pair. The resultToString()method prints the results to users for viewing.

5.5 Business Application Interpretation

The prototype system implements several formats of property result interpretation. For the online interpretation, SNA property searching results can be interpreted to a Titan graph, in a matrix, or as a list. For the offline interpretation, the system supports matrix and list format.

Figure 5.9 shows the class diagram for the property result interpreter component. Users choose one or several properties and a specific interpreter class from the PropertyInterpreterFactory for a certain format. There are two interpreter interfaces: the

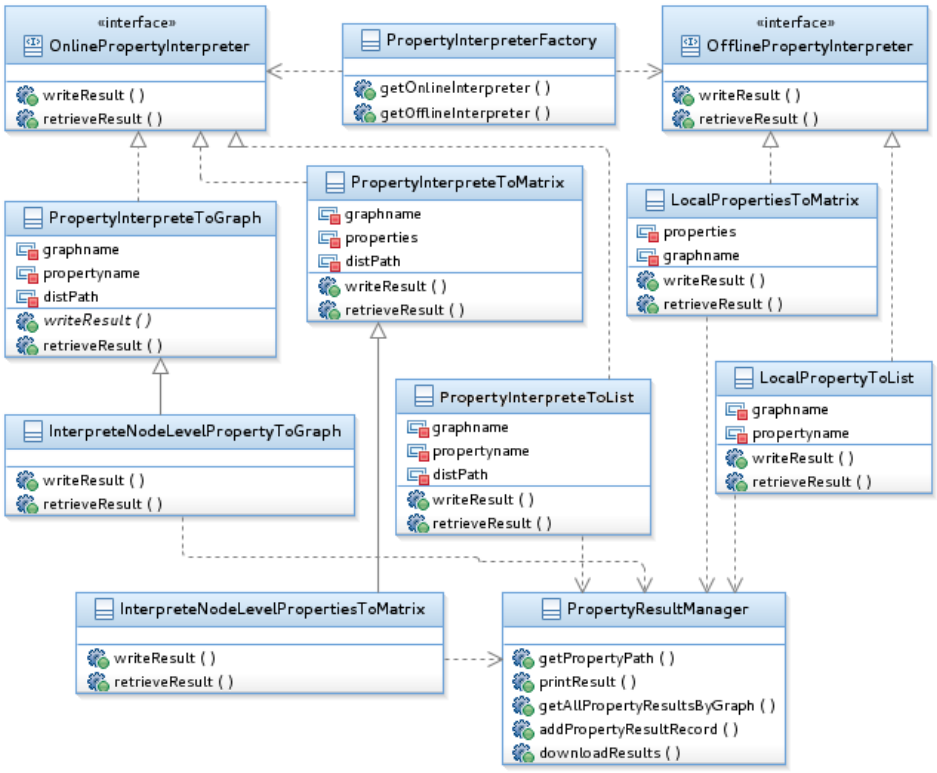


Figure 5.9: The Class Diagram of Property Result Interpreter Component

OnlinePropertyInterpreter and OfflinePropertyInterpreter. The OnlinePropertyInterpreter is for processing property results in the repository, and the OfflinePropertyInterpreter is for the downloadable property results. They both have two methods: writeResult() and retrieveResult(). Users need to call writeResult() method at first to convert the results and then use retrieveResult() method to retrieve results.

There are three classes implementing the OnlinePropertyInterpreter, as each class interprets to a corresponding format, i.e. graph, matrix and list. The InterpreteNodeLevelPropertyToGraph writes the node level property results as the vertices' attributes to the graph in Titan, and returns a Titan graph configuration to the user. The InterpreteNodeLevelPropertiesToMatrix combines several node level properties such as centralities to a matrix. The PropertyInterpreteToList retrieves the list format to users.

The offline interpreter runs the process in local tasks. The PropertyResultManager is for downloading the results. The LocalPropertiesToMatrix keeps a vertex list in memory and writes to a matrix file.

Chapter 6

Case Study

In this chapter, we describe a case study to show business applications of the system. The case study is for identifying key players in the stock network by using SNA centralities and comparing the changes in different time periods. The case study is based on two business research papers and uses the proposed SNA methodologies in the papers [38] [37].

In the stock market, the fluctuations of prices of a stock are potentially related to each other. Analyzing stock on network is proposed to study the influences and correlations of stocks according to the stock prices [49]. In a stock network, stocks are nodes, and edges are created presenting the correlations between the stocks. The correlations are based on the stock prices as a measurement of the similarities between two stocks.

To construct stock networks and determine the significance of each stock, this case refers to the work in [38], which constructed a stock correlation network and used traditional centralities to measure the network: degree, betweenness, closeness and eigenvector centralities. Finally, the paper used the PCA method to combine the four centralities and summarize an overall score of each stock. In [37], the authors built stock networks in different time periods and examine the evolutional changes

of the networks. Combining the methodologies in these two papers, this case study was trying to find influential stock nodes and compared the network changes along the time line. Next, we explain the analytic steps using the system.

6.1 Data Preparation and Collection

The stock networks are built on the 500 stocks in NYSE. The original stock files contains an entire range of daily price from 2008 Jan to 2012 Jan obtained from Yahoo Finance, as shown in Figure 6.1. Before submitting the source data files to the system, the user should transfer the original files to time sequence format that the system accepts. Each file in the time sequence format should at least include the data unit's identifier, start time, end time, time interval and time series data. Figure 6.2 is an example of the stock BMO in the time sequence format. The user chooses the abbreviation of each stock as the identifier (BMO). The start time (2008-1-4) and end time (2013-1-3) can be extracted from the file. The time interval is daily. For building stock network, the cross correlation is calculated by the logarithm of the price of the stock. To get useful results, the adjusted closing price is chosed to calculate as it is derived by amending the closing price for any stock splits, dividends and distributions. The logarithm of the price of stock i at day t is:

$$r_i(t) = \ln p_i(t) - \ln p_i(t-1) \tag{6.1}$$

where $p_i(t)$ is the adjusted closing price.

After the files are preprocessed to the time sequence format shown in Figure 6.2, the user submits the file to the repository.

Prices						
Date	Open	High	Low	Close	Volume	Adj Close*
Dec 30, 2011	54.38	55.03	54.38	54.81	1,512,000	45.50
Dec 29, 2011	54.17	54.43	54.06	54.38	993,100	45.14
Dec 28, 2011	54.77	54.85	53.92	54.05	676,700	44.86
Dec 27, 2011	54.45	54.62	54.21	54.26	332,100	45.04

Figure 6.1: An example of Stock File from Yahoo Finance

```
#name,BMO
#startTime,2008-1-4
#endTime,2013-1-3
#timeInterval,daily

2013-01-03,-1.6287971369166598E-4
2013-01-02,0.012949939189902615
2012-12-31,0.005127792230053885
2012-12-28,-0.0036417851870149676
2012-12-27,-0.00428689856849207
2012-12-26,-0.0027930684386339166
```

Figure 6.2: An example of the Time Sequence File

6.2 Network Construction

The stock network is constructed by calculating the correlations for each pair of stocks based on their time series of stock prices and connected the stocks according to their cross correlations. Based on (6.1), the correlation is defined as:

$$c_{ij} = \frac{\langle r_i r_j \rangle - \langle r_i \rangle \langle r_j \rangle}{\sqrt{(\langle r_i^2 \rangle) - \langle r_i \rangle^2)(\langle r_j^2 \rangle) - \langle r_i \rangle^2)}} \tag{6.2}$$

The network is usually large which makes the analytic work complex. To filter the important information and reduce the complexity, [38] refers to minimum spanning tree (MST) which forms a simpler topological structure of the network. However, essential information of the network is inevitably lost by using MST to filter correlation networks. To fit the MST topological criterion, dramatic edges of high correlations are removed while edges of low correlations are retained [49]. Another limitation of

the MST method is that it does not consider the situation that the system contains more than one MST. Therefore, instead of using MST, we use the threshold to filter the correlation. The edges of the two stocks are determined to be created if their cross correlation is larger than a threshold (e.g. 0.5).

The user builds networks by giving graph generation specification to the system. According to the network construction method discussed above, Table 6.1 is a standard specification for our case. Although the system lets the user decide the graph name, it is the identifier of the graph and should be unique. In the example, the graph name is recommended in the form of 'marketname_timeperiod_R function and threshold', which represents the network building information and guarantee the uniqueness. The graph is defined as a weighted undirected graph. When setting the time period, as Paper [37] stated, according to the random matrix theory (RMT), when the number nodes n in a network is larger enough, the noise of correlation matrix within finite time period T is greatly reduced if $Q = (T/n) \geq 1$. In our case, there are 500 nodes, so the time period of the daily price should be larger than 500 days. The selected time period in the example is from 2008-1-4 to 2009-12-31, which contains 504 days of trading prices. According to the formula 6.2, the R function is Pearson Correlation. To measure the correlation strength, a value larger than 0.5 is considered a moderate positive relationship, so the threshold is set from 0.5 to 1 [50]. As the user wants to set the correlation values as weights, the weighted method is none.

6.3 SNA Property Calculation

After the graph is created, we use four centralities to determine the relative importance of the stocks: degree centrality, betweenness centrality, closeness centrality and eigenvector centrality. First the degree centrality of stock i is defined as [51]:

Table 6.1: Example for Graph Generation Specification

graphname= NYSE500_20080104_20091231_pearsonCor_0.5_1
sourcedir=nyse_500_timeseq
graphdirection=false
graphweight=true
inputformat=timeSequenceInputFormat
starttime=2008/1/4
endtime=2009/12/31
func.Rfunc=pearsonCorrelation
func.result.threshold=0.5:1
func.result.weightmethod=none

$$DC(i) = \frac{\sum_{j=1}^{k} A_{ij}}{N-1} \tag{6.3}$$

where A_{ij} is the weight between stocks i and its neighbor stocks, and N is the total number of stocks. The betweenness centrality of stock i is [51]:

$$BC(i) = \sum_{j,k \in V} \frac{\sigma_{jk}(i)}{(N-1)(N-2)\sigma_{jk}} \tag{6.4}$$

where $\sigma_{jk}(i)$ denotes the weighted sum of the edges along the paths between any pair (j,k) in the network going through stock i. σ_{jk} denotes the weighted sum of the shortest path between stock j and stock k. The $(N-1)(N-2)$ is divided for normalization.

After filtered by the correlation threshold, the network is very likely to be formed by disconnected components, and nodes in small components are generally closer to each other compared to larger components, which leads to much larger closeness centrality values if we use the Freeman's traditional closeness centrality measurement. To overcome this issue, we refer to Opsahl's closeness algorithm [52], which is:

$$CC(i) = \sum_{j} \frac{1}{(N-1)d_{ij}} \tag{6.5}$$

where the equation is the weighted sum of inverse distance instead of the inverse

of the sum of distance and normalized as the closeness values. The eigenvector centrality of stock i is defined as [51]:

$$EC(i) = \frac{1}{\lambda_{max}} \sum_{j=1} A_{ij} x_j \tag{6.6}$$

Each eigenvalue is normalized by dividing it by the largest eigenvalue λ_{max} in the network.

6.4 Result Retrieving and Interpretation

After the property calculations are finished, each stock has four centrality values which measure the influences of the stocks in different perspectives. Based on the matrix, we apply the PCA method to get an overall score for each stock. The matrix data can be retrieved from the system. The matrix is $N \times 4$ size, as each row contains four columns representing four centrality values in order: betweenness, closeness, degree and eigenvector centrality. Then we get the covariance matrix of the original matrix, and calculates eigenvectors and eigenvalues of the covariance matrix. We represent the eigenvector with the largest eigenvalue as $v = (v_1, v_2, v_3, v_4)$, which is 4×1 size. For the stock i, its overall score is:

$$O_i = DC_i * v_1 + BC_i * v_2 + CC_i * v_3 + EC_i * v_4 \tag{6.7}$$

The overall scores is then used to give a new rank of the stocks to identify significant stocks.

6.5 Network Comparison

In this case study, we iteratively build stock networks for different time periods. Referring to the method in [37], the dataset is divided into 27 periods from 2008

Jan to 2011 Dec. The length of each time period is 504 trading days. A series of consecutive periods are obtained by sliding the time window of 4 weeks (28 exchange days). Period 28 has a full length of data from 2008-1-4 to 2011-12-29. For each period, a graph is built with the same threshold value (0.5). We examine the numbers of nodes and edges changes of each graph. We also compare the changes of the top stocks in the ranks of different periods.

Chapter 7

Experiments

In this chapter, based on the case study, we report the experimental details and results by using the system.

7.1 Experimental Environment

The experiment for the case study is run on a Hadoop cluster with 13 machines, where one machine is the server and master, and others are the slave workers. The server is the head of Hadoop and HBase, as it is Hadoop's NameNode, ResourceManage and HBase's Master at the same time. The Titan packages and the system's main packages are located on the server as well. The 12 slave workers are Hadoop's DataNode, NodeManager and HBase's RegionServer. For this case, we also built a web-based interface to do analytic work. The web application uses Tomcat as the web server and use Spring for forwarding the requests. Users can interact with the system through web interfaces for use activities.

7.2 Experimental Result

7.2.1 Centrality Measures

Table 7.1 is the matrix file output by the system, which is based on the graph whose threshold is 0.5 and time period is from 2008-1-4 to 2011-12-29. It reports the centrality scores of the stocks in the graph. Data in the third column to the sixth column are betweenness, closeness, degree, eigenvector centrality and the overall score. The graph has 457 stock nodes, so the matrix is 457×4 size.

Table 7.1: Matrix File

No.	Stock	BC	CC	DC	EC
1	HNI	0.001328	1.345726	0.264054	0.598772
2	CDE	2.60E-04	0.945871	0.025705	0.053169
3	XOXO	0	0.98508	0.021347	0.062002
4	CDR	4.05E-05	1.128793	0.105485	0.279542
5	GLW	6.21E-04	1.343571	0.251706	0.621084
6	DFT	0	0.838374	0.001132	0.003407
7	WLL	0.001796	1.277274	0.262018	0.594323
8	CEC	2.96E-04	1.156534	0.122456	0.305555
9	AA	0.00135	1.424398	0.375527	0.81233
10	UHT	0.002017	1.405147	0.33984	0.711372
...		...	...	...	
455	TEL	8.18E-04	1.445575	0.335832	0.757308
456	RAI	0.003332	0.895661	0.006939	0.013581
457	TEO	0	0.895169	0.005564	0.01917

We rank the stocks according to each centrality scores as shown in Table 7.2. We identify the top ones with higher scores for each centrality. Betweenness centrality states the probability of a stock lying in the shortest path of any two stocks in the network. A stock with a higher betweenness centrality plays more important roles to coordinate with the other stocks in the network [38]. Table 7.2a shows the top five stocks with the highest betweenness centrality scores. The stock with the highest betweenness score is AMG, following by K, NJR, IEX and CR.

Closeness centrality measures the distance of a stock to other stocks in the network.

A stock with a higher closeness centrality spreads its influence more quickly to the rest of the network. Table 7.2b shows the top five closeness centrality, which is IEX, ITW, EV,HON and PNR.

Degree centrality calculates the number of edges. Stocks with higher degree centrality will influence more stocks directly [38]. Table 7.2c shows the top five stocks with the highest degree centrality scores, which are DD, PNR, IEX, EV and HON.

Eigenvector centrality measures the importance of a stock according to its neighbors. If a stock connects to the stocks which have high scores, it will also have a high score. In Table 7.2d, the top five stocks with the highest eigenvector centrality are DD, PNR, HSC, HON and ITW.

According to the top five ranks of the four centralities, IE and PNR appear three times. DD, ITW, HON and EV appear two times.

As described in the case study, to determine the most influential stocks, an overall score is defined by combing the four centralities using PCA. Here R is used to do the PCA calculation. We call R's prncomp function with the matrix as the input. In the PCA, the first principle accounts for 79.6% of the total variances which is sufficient to determine the overall scores [24]. The vector of the first principal component is $v = (v_1, v_2, v_3, v_4)$, where $v_1 = 0.0018, v_2 = 0.5664, v_3 = 0.3579, v_4 = 0.7424$. It shows that in this network, eigenvector centrality is the most important measure related to the covariance structure, which is followed by closeness centrality, degree centrality and betweenness centrality. The overall scores are calculated by using these values in equation (6.7). We ranked the stocks with the overall scores and listed with the traditional four centralities, as shown in Table 7.3. The top six stocks with the highest overall scores are DD, PNR, IEX, ITW, HON and EV. All the top six stocks appear two to three times in the top five lists for the four individual centralities.

Figure 7.1 visualizes the graph in Gephi. As we can see, the graph has a giant component in which nodes are tightly knitted together. Nodes in the central cluster are

Table 7.2: The Top Five Stocks with the Higher Centralties

(a) Betweenness Centrality

Stock	BB
AMG	0.009363
K	0.007024
NJR	0.006683
IEX	0.006292
CR	0.006216

(b) Closeness Centrality

Stock	CC
IEX	1.556973
ITW	1.556781
EV	1.546406
HON	1.541396
PNR	1.540632

(c) Degree Centrality

Stock	DC
DD	0.53653
PNR	0.501962
IEX	0.500194
EV	0.498713
HON	0.49686

(d) Eigenvector Centrality

Stock	EC
DD	1
PNR	0.957992
HSC	0.954036
HON	0.950244
ITW	0.946394

Table 7.3: Centrality Scores

Rank	Stock	BC	CC	DC	EC	O
1	DD	0.001515	1.53171	0.53653	1	1.80194
2	PNR	0.001455	1.540632	0.501962	0.957992	1.76343
3	IEX	0.006292	1.556973	0.500194	0.946282	1.76337
4	ITW	0.002567	1.556781	0.496663	0.946394	1.76208
5	HON	7.24E-04	1.541396	0.49686	0.950244	1.75629
6	EV	0.003677	1.546406	0.498713	0.944306	1.75539
7	HSC	0.003521	1.531225	0.494814	0.954036	1.75261
8	RPM	0.002301	1.532791	0.489164	0.943058	1.74333
9	ROP	0.001454	1.533358	0.485202	0.94027	1.74016
10	KMT	0.002786	1.515321	0.482631	0.94143	1.72989
...	...	...	...	...	..	.
454	KR	0	0.00318	0	6.63E-229	0.0018
455	SWY	0	0.00318	0	6.63E-229	0.0018
456	NTT	0	0.003067	0	2.50E-227	0.00174
457	DCM	0	0.003067	0	2.50E-227	0.00174

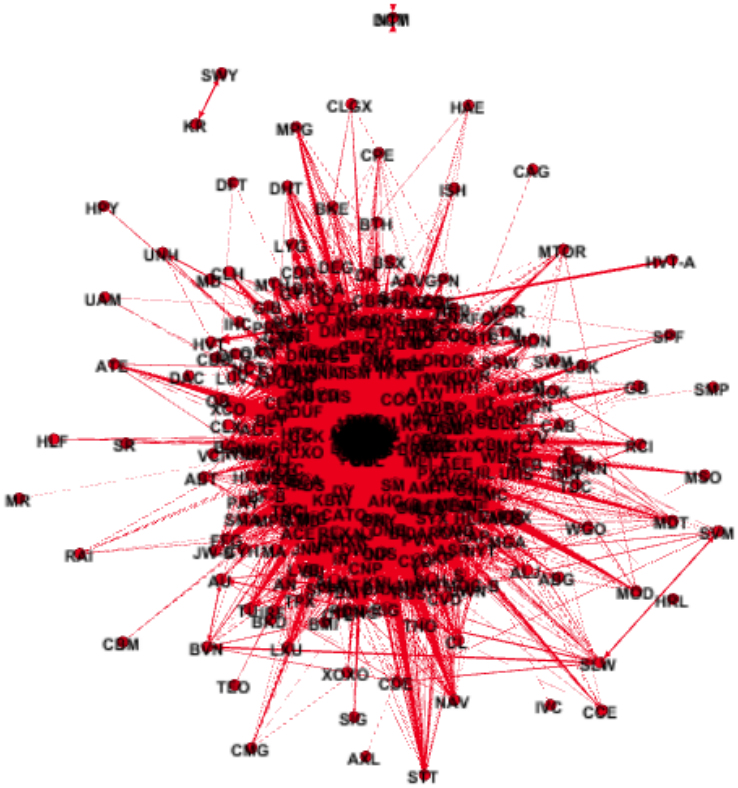


Figure 7.1: The Stock Network Structrue of the Graph in Period 1

very dense. The score rank helps determine key nodes which cannot be identified from viewing the graph. There are two small components at the periphery. Accordingly, the nodes (SWY, KR, DCM) in the two components are listed at the last of the rank, which verifies that the score rank indeed captures the centrality property in the network.

7.2.2 Network Comparisons

We built stocks networks and measured the centralities of the stocks in different time periods between 2008-1-4 and 2011-12-29. Such stock networks can reflect the impact of financial crises on the companies within that period. Referring to the work in [37], we study the network behaviors before and after the collapse of Lehman Brothers. We compared the numbers of the nodes and edges evolutions, and their rank changes between different networks. As mentioned in the last chapter, the datasets are divided into 28 periods. The time window of a period is 504 exchange days. Period 1 to 27 are obtained by sliding the time window for a month data points (20 exchange days). Period 28 includes the entire data of 1006 exchange days. Table 7.4 shows the start date and end date of some selected periods.

Table 7.4: Sample Time Periods

Period	Start Date	End Date
1	2008-1-4	2009-12-31
2	2008-1-31	2010-1-28
11	2008-10-9	2010-10-7
12	2008-11-6	2010-11-4
21	2009-7-16	2011-7-14
22	2009-8-13	2011-8-11
27	2009-12-31	2011-12-29
28	2008-1-4	2011-12-29

We calculated the nodes and edges of the network in each period and investigated the changes, as shown in Figure 7.2. The number of edges increased in a short term before the Period 11 and then suffered a sharp decrease after that, which is a

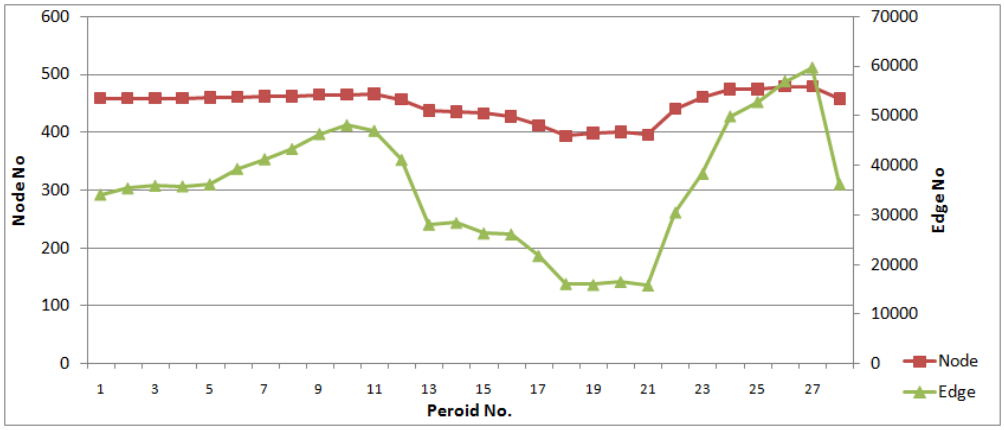


Figure 7.2: Evolution of the Numbers of Nodes and Edges

synchronized reflection of the financial crisis. The edges numbers stayed low after the collapse of the Lehman Brothers, which indicated that less stocks are connected to each other in the network. From Period 21 to Period 27, edges gradually increased leading to the stock network with relative high density, which shows stokes in the networks had more strong correlations to each other. Period 28 measures the network data in the entire period. The numbers of the nodes followed a similar fluctuation.

We apply the centrality measures described previous on the networks in the 28 periods, then get a rank of the scores for each network according to the overall centrality. Table 7.5 selectively shows the top ten stocks in some periods. In Period 10, before the numbers of nodes and edges began to decreased, we can see that the top stocks had been different from the former periods. In Period 14, the top stocks changed again as some new stocks emerged in the rank. The stocks in the rank of Period 21 to Period 22 were shuffled, and the two periods were also very unlike as well, which showed the networks were very unstable. In the last periods, the top stocks changed again, but the changing pace started to slow down. From the top ranks, we can identify some significant stocks such as DD, HON and PNR. Although the networks were changing, they still appeared in multiple times.

Besides the top stocks, we also observe that the range of the stock rank changes.

Table 7.5: The Top Ranks Over Some Periods

Rank	P1	P10	P14	P21	P22	P27	P28
1	DD	RPM	HON	AFG	HON	IEX	DD
2	HSC	DD	DD	GDI	LUK	HON	PNR
3	EV	PNR	RPM	ROP	IEX	PNR	IEX
4	RPM	HSC	SE	KMT	ROP	EV	ITW
5	DIS	PH	AEG	EMN	PNR	LUK	HON
6	PNR	EV	LUK	JNS	SE	DD	EV
7	ITW	HON	EMN	MWV	TOT	AFG	HSC
8	ROP	IEX	CVX	CVX	IFF	CR	RPM
9	HON	SE	PNR	TDY	DD	KMT	ROP
10	AVY	KMT	AMG	ALV	EMN	AMG	KMT

For the stocks which appear in both networks of Period 1 and 10, the biggest jump is from 103 to 268, and the range of the changes is 258. The range of changes from Period 10 to Period 11 is 149, and range from Period 21 to 22 is 377 which is a relatively high change. It shows that the networks were very unstable between these periods.

Chapter 8

Conclusion

8.1 Summary

In this thesis, we propose a scalable analytics system built for using social network analysis approach on large business datasets. The system's goal is to help business researchers analyze large volume of data more effectively and efficiently. The system supports five major functions: data collection, graph generation, graph reuse, property searching and business application interpretation. We classify business data formats according to a number of business application research papers and define two data formats for graph generation: time sequence format and transaction list format. For each data format, the system can build SNA graphs from source data automatically. The system can also generate graphs from existing graphs for SNA property search. The system has several built-in SNA property search methods that users can choose to compute. The SNA property results are saved and can be retrieved by users' applications. The system facilitates several result formats for property result interpretation.

In the system design, we first give the required system features. Based on the features, we design the system architecture which consists of six key components:

source parser, graph loader, graph reuser, network analyzer, property results interpreter and graph visualizer. In the implementation, we describe the strategy and how the components cooperate between components. Then we implement a prototype system using the Hadoop cluster. The system integrates a number of big data analytics tools and graph analysis techniques to manage graph data and detect SNA properties, including Hadoop, Titan, Giraph and R.

In the case study, we demonstrate how the system can help analyze stock networks, and identify key players with the centrality results. The case study also compares stock network properties in different time periods. The analytic approaches are from the relevant business application research papers. The experimental results show that the system is capable of building, storing and analyzing large-scale graphs to achieve the research goals. It reduces manual graph building work, simplifies the SNA process, improves data processing efficiency and makes it easier to work with large data sets.

8.2 Contribution

First, we propose a general-purpose software system that combines SNA and big data techniques for business data. To our knowledge, there is no similar system currently available. There are a number of existing projects and tools for SNA, such as GraphBuilder, Pajek, JUNG and Commetrix, but most projects are not for big data. Projects like Pregel and GraphX support large graph processing but are not for graph building and storage. Our system supports a complete workflow from graph construction to graph analysis operating on large-scale data.

Second, we design a software architecture based on Hadoop, and we also define the data structure in the repository which stores multiple types of data. We define virtual graph and physical graph in the system and design a set of graph reuse approaches

for different data formats.

Third, we implement a prototype system by integrating several big data systems, such as Hadoop, HBase and Titan. In the backend, the system coordinates with different formats of data from different big data systems. Our system lets users focus on their analytics work at the high level.

8.3 Future Work

There are still parts in the system which are needed to be improved in future.

First, the strategies adopted in our system are in an early stage of development. We define two data formats for graph generation according to the existing works which do not cover all the situations. As more research projects with modeling graphs appear, we can add more supporting data formats and graph generating methods to cover a wider range of data formats.

Second, the main part of the system is based on combining different existing big data systems which results in restricting the system's functions. For example, the graph database Titan does not support graph mutation physically which makes the graph reuse processing very painful. Moreover it is not most efficient because of data conversion problem from one big data system to another system. For example Giraph API can not read graph directly from Titan in the current version. We must create some intermediate files to integrate them. We need a better way is to develop APIs to let the projects cooperate with each other directly.

Third, we do not consider about security issue in the system so far. In the future, users should be only capable of calculating and viewing the graphs and properties that they owned.

Bibliography

[1] A. Hadoop, "Hadoop architecture." [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[2] ——, "Apache yarn architecture." [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[3] "Big data," Apr. 2016, page Version ID: 715370718. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Big_data&oldid=715370718

[4] "Why Big Data is the new competitive advantage." [Online]. Available: http://iveybusinessjournal.com/publication/why-big-data-is-the-new-competitive-advantage/

[5] "What is big data analytics? - Definition from WhatIs.com." [Online]. Available: http://searchbusinessanalytics.techtarget.com/definition/big-data-analytics

[6] G. U. (2001) Social network analysis: Introduction and resources. [Online]. Available: http://lrs.ed.uiuc.edu/tse-portal/analysis/social-network-analysis/

[7] "Using Big Data in Social Network Analysis," Jul. 2013. [Online]. Available: http://offspringsolutions.com/info/using-big-data-in-social-network-analysis/

[8] "Social network," Apr. 2016, page Version ID: 714627944. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Social_network&oldid=714627944

[9] "Social network analysis: An introduction." [Online]. Available: http://www.orgnet.com/sna.html

[10] S. P. Borgatti, A. Mehra, D. J. Brass, and G. Labianca, "Network analysis in the social sciences," *Science (New York, N.Y.)*, vol. 323, no. 5916, pp. 892–895, Feb. 2009.

[11] K. Fuks, A. Kawa, and B. Pierański, "Adaptation of social network analysis to electronic freight exchange," in *New Trends in Intelligent Information and Database Systems*. Springer, 2015, pp. 151–159.

[12] "Graph theory." [Online]. Available: http://webwhompers.com/graph-theory.html

[13] Hadoop. (2015) Apache hadoop. [Online]. Available: http://hadoop.apache.org/

[14] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[15] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[16] H. Wiki, "Powered by apache hadoop." [Online]. Available: http://wiki.apache.org/hadoop/PoweredBy

[17] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, Inc., 2013.

[18] Aurelius. (2015) Titan the distributed database. [Online]. Available: http://thinkaurelius.github.io/titan/

[19] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[20] Giraph. (2015) Welcome to apache giraph! [Online]. Available: http://giraph.apache.org

[21] Spark. (2015) Graphx programming guide. [Online]. Available: http://spark.apache.org/docs/latest/graphx-programming-guide.html

[22] Aurelius. (2015) Titan with tinkerpops hadoop-gremlin. [Online]. Available: http://spark.apache.org/docs/latest/graphx-programming-guide.html

[23] W. D. 50 shades of graph: How graph databases are transforming online dating. [Online]. Available: http://www.forbes.com/sites/danwoods/2014/02/14/50-shades-of-graph-how-graph-databases-are-transforming-online-dating/

[24] Y. Kim, T. Y. Choi, T. Yan, and K. Dooley, "Structural investigation of supply networks: A social network analysis approach," *Journal of Operations Management*, vol. 29, no. 3, pp. 194–211, 2011.

[25] T. Y. Choi and Y. Hong, "Unveiling the structure of supply networks: case studies in honda, acura, and daimlerchrysler," *Journal of Operations Management*, vol. 20, no. 5, pp. 469–493, 2002.

[26] G. Martino, F. M. Sarti, and F. Panella, "Social network analysis in encouraging role-players in the beef market to take breeding decisions: A methodological study," *Italian Journal of Animal Science*, vol. 12, no. 1, p. e9, 2013.

[27] A. T. Stephen and O. Toubia, "Deriving value from social commerce networks," *Journal of marketing research*, vol. 47, no. 2, pp. 215–228, 2010.

[28] C. Fieseler, M. Fleck, and M. Meckel, "Corporate social responsibility in the blogosphere," *Journal of Business Ethics*, vol. 91, no. 4, pp. 599–614, 2010.

[29] O. Levina and A. Bobrik, "Using social network analysis to measure information management performance introduced by business process optimization." in *ECIS*, 2013, p. 111.

[30] S. Palus, P. Bródka, and P. Kazienko, "How to analyze company using social network?" in *Knowledge Management, Information Systems, E-Learning, and Sustainability Research*. Springer, 2010, pp. 159–164.

[31] P. Gloor, P. Dorsaz, and H. Fuehres, "Analyzing success of startup entrepreneurs by measuring their social network distance to a business networking hub," in *Proceedings 3rd international conference on collaborative innovation networks coins, Basel. Sept*, 2011, pp. 8–10.

[32] E. Giuliani and C. Pietrobelli, "Social network analysis methodologies for the evaluation of cluster development programs," Inter-American Development Bank, Tech. Rep., 2011.

[33] C. van Egeraat and D. Curran, "Social network analysis of the irish biotech industry: implications for digital ecosystems," in *Digital Ecosystems*. Springer, 2010, pp. 31–43.

[34] J. L. Miller, "Producing quality: A social network analysis of coproduction relationships in high grossing versus highly lauded films in the us market," *International Journal of Communication*, vol. 5, p. 20, 2011.

[35] M. Rancan, "The Value of Social Networks in Financial Markets," European University Institute, RSCAS Working Papers 2013/21, Mar. 2013. [Online]. Available: https://ideas.repec.org/p/rsc/rsceui/2013-21.html

[36] S. L. Siregar, D. C. Pardede, and R. S. Wahyuni, "Structural positions and financial performances of rural banks in central java network (cj-net): A social network analysis perspective on apex-rural banks scheme," *International Proceedings of Economics Development and Research*, vol. 84, p. 6, 2015.

[37] R. B. Roy and U. K. Sarkar, "Identifying influential stock indices from global stock markets: A social network analysis approach," *Procedia Computer Science*, vol. 5, pp. 442–449, 2011.

[38] G. S. Lee and M. A. Djauhari, "An overall centrality measure: The case of us stock market," *International Journal of Electrical & Computer Sciences*, vol. 12, no. 6, 2012.

[39] T. Raeder and N. V. Chawla, "Market basket analysis with networks," *Social network analysis and mining*, vol. 1, no. 2, pp. 97–113, 2011.

[40] I. F. Videla-Cavieres and S. A. Ríos, "Extending market basket analysis with graph mining techniques: A real case," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1928–1936, 2014.

[41] V. Tomar, H. Asnani, A. Karandikar, V. Chander, S. Agrawal, and P. Kapadia, "Social network analysis of the short message service," in *The 16th National Conference on Communications (NCC 2010). Chennai, India: IEEE.* Citeseer, 2010, pp. 1–5.

[42] C. A. R. Pinheiro, "Highlighting unusual behavior in insurance based on social network analysis," in *Community Detection to Identify Fraud Events in Telecommunications Networks.* SAS Global Forum Proceedings, 2012, p. 106.

[43] W. Zeng, Y. Huang, and L. Jiang, "The study of microblog marketing based on social network analysis," in *Information Management, Innovation Management*

and Industrial Engineering (ICIII), 2011 International Conference on, vol. 3. IEEE, 2011, pp. 410–415.

[44] "Harnessing Hashtagify.me and social network analysis to maximize your brands reach on Twitter," Mar. 2014. [Online]. Available: http://www.randalolson.com/2014/03/05/harnessing-hashtagify-me-and-social-network-analysis-to-maximize-your-brands-reach-on-twitter/

[45] J. Zhang, D. Du, and W. Du, "A software toolkit for stock data analysis using social network analysis approach." The 2014 International Conference on Advances in Big Data Analytics.

[46] Google, "Designing Your Schema." [Online]. Available: https://cloud.google.com/bigtable/docs/schema-design

[47] D. M. Shook, Adam, *MapReduce Design Patterns*.

[48] "tinkerpop/gremlin." [Online]. Available: https://github.com/tinkerpop/gremlin

[49] C. K. Tse, J. Liu, and F. C. M. Lau, "A network perspective of the stock market," *Journal of Empirical Finance*, vol. 17, no. 4, pp. 659–667, Sep. 2010.

[50] D. J. Rumsey, *Statistics For Dummies*, 2nd ed. Hoboken, N.J: For Dummies, May 2011.

[51] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, Jan. 1978.

[52] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Social Networks*, vol. 32, no. 3, pp. 245–251, 2010.

Vita

Candidate's full name: Fan Liang

Universities attended:
Guangxi University, China
September 2007 - July 2012
Bachelor of Management

Southeast University, China
Septemper 2012 - July 2014
Master of Software Engineering

Poster:
Fan Liang and Weichang Du, "Poster: Big Data Analytics Toolkit for Business Data," UNB Computer Science 2016 Research Exposition, April 8, 2016, Fredericton, NB, Canada