THESIS

DETERMINING DISEASE OUTBREAK INFLUENCE FROM VOLUMINOUS EPIDEMIOLOGY DATA ON ENHANCED DISTRIBUTED GRAPH-PARALLEL SYSTEM

Submitted by

Naman Shah

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2017

Master's Committee:

Advisor: Sangmi Lee Pallickara

Shrideep Pallickara
Daniel E Turk

Abstract

DETERMINING DISEASE OUTBREAK INFLUENCE FROM VOLUMINOUS EPIDEMIOLOGY DATA ON ENHANCED DISTRIBUTED GRAPH-PARALLEL SYSTEM

Historically, catastrophe has resulted from large-scale epidemiological outbreaks in livestock populations. Efforts to prepare for these inevitable disasters are critical, and these efforts primarily involve the efficient use of limited available resources. Therefore, determining the relative influence of the entities involved in large-scale outbreaks is mandatory. Planning for outbreaks often involves executing compute-intensive disease spread simulations. To capture the probabilities of various outcomes, these simulations are executed several times over a collection of representative input scenarios, producing voluminous data. The resulting datasets contain valuable insights, including sequences of events that lead to extreme outbreaks. However, discovering and leveraging such information is also computationally expensive.

This thesis proposes a distributed approach for aggregating and analyzing voluminous epidemiology data to determine the influential measure of the entities in a disease outbreak using the PageRank algorithm. Using the Disease Transmission Network (DTN) established in this research, planners or analysts can accomplish effective allocation of limited resources, such as vaccinations and field personnel, by observing the relative influential measure of the entities. To improve the performance of the analysis execution pipeline, an extension to the Apache Spark GraphX distributed graph-parallel system has been proposed.

Acknowledgments

First and foremost, I would like to thank God for his kindness and grace before and during this thesis. I believe He offered me perfect opportunities and made me walk on the avenue of right choices, which led me to this endeavor.

I would like to thank my family, my mother, Dr. Dipika Shah, my father, Rajiv Shah, and my little sister, Khushi Shah, for their infinite love and strong support in every stage of my life. I am thankful that they allowed me to pursue the Master's degree, which made this work possible.

I am heartily grateful to my advisor, Dr. Sangmi Lee Pallickara, for believing in me. Her insightful guidance and continuous encouragement during the dark days of this study made it a successful attempt. Dr. Pallickara, like you mentioned at the beginning of this research, this was an unforgettable experience for my lifetime. I would like to express my gratitude to my committee members, Dr. Shrideep Pallickara and Dr. Daniel E Turk, for their valuable inputs and prudent advice during this work. I am thankful to the whole committee for their patience & cooperation in completing this work. I would like to acknowledge the US Department of Homeland Security and the US National Science Foundation, for funding this research.

I would like to thank my ingrained friends, Nikhil Patel, Mital Patel, Aman Shah, Ketur Patel, Jainesh Patel, Nishant Patel and Umang Mehta, and my cousin, Madhavi Shah, in no particular order, for their immense support and motivation towards achieving this goal. I shared my failures and successes with them. I am especially grateful to them for taking care of my family in my absence. Many thanks to Pinal Patel and his wife, Ankita Patel, for being a home away from home in this foreign land. Thank you, Pinal, for your presence at my defense, and arranging those refreshments. I would like to thank other known families in Fort Collins, especially Mr. Dipen Patel & family, for inviting me over to social events and helping me in getting familiarized with this town. Thank you, Dipenbhai, for attending my defense.

I would like to thank Harshil Shah and Dr. Matthew Malensek for their help and support in publishing this research. Greatest thanks to Johnson C K Arulswamy, for being a supportive colleague, for those long discussions regarding problem solutions, for all the technical help, and reviewing this manuscript. Jo, I am going to miss those discussions. I would like to express my gratitude to all the Graduate Research Assistants from Big Data Group as well as Distributed Systems Group, for their presence at my defense.

Last but not least, I would like to thank Dr. Deborah Parker, for proofreading and editing this document on a short notice.

This thesis is typset in LaTeX using a document class designed by Leif Anderson.

Dedication

To my family

Table of Contents

List of Figures

CHAPTER 1

Introduction

According to the Food and Agriculture Organization (FAO), there are currently more than 1.5 billion cattle, 1.1 billion sheep, and 0.97 billion pigs and goats in the global livestock industry, which employs at least 1.3 billion people [1]. Diseases in livestock have serious effects on ecological systems, the global economy, and human health in the case of zoonotic diseases, such as Ebola and swine flu, which can be transmitted from animals to humans. In 2001, the Foot and Mouth Disease (FMD) outbreak in the United Kingdom infected nearly 10 million livestock animals, and approximately 4 million of them were slaughtered during the process of disease control. In addition, this outbreak affected agriculture, rural tourism, and food supply chains, resulting in overall damage to the economy of the country [2]. It has been estimated that yearly direct production losses and vaccination costs due to highly contagious FMD are in the range of US$6.5 billion to $21 billion in infected regions, compared to just over US$1.5 billion in FMD free regions [3].

One of the main challenges during an outbreak is the lack of available resources essential to minimizing the catastrophe, such as skilled veterinarians, veterinary stockpiles, and quarantine stations, that are essential to minimize the catastrophe. Also, the number of trained veterinarians has been declining for the last two decades, and it is anticipated that this is likely to lead to a significant gap between supply and demand [4]. Moreover, the U.S. Department of Agriculture - Animal and Plant Health Inspection Service (USDA-APHIS) manages the National Veterinary Stockpile (NVS) program, which maintains and supplies critical veterinary stockpiles consisting of vaccines, antivirals, and drugs. This organization delivers stockpiled supplies to infected areas within 24 hours from a request by the state in which the outbreak is occurring [5]. Given an outbreak of FMD, clinical signs of the disease appear at least 2 to 3 days after infection [6], during which, other animals and humans can be infected by means of mediums the include direct or indirect contact, shipments, and airborne spread. This timespan, along with the issue of availability of the

veterinary stockpile and limited supply of veterinarians, easily create conditions that allow a normal infection chain to become an outbreak. With this set of obstacles, the ideal approach to prevent a disease outbreak is to use resources efficiently by targeting those influential premises that affect others disproportionately. In other words, once a particular influential premise is infected, the properties of disease spread; for example, overall disease duration and the total number of premises infected are high. Studies have shown that strategic vaccination is certainly more effective than its counterpart, the random approach [7]. The same notion can be applied to effective resource utilization. Hence, determining influence of premises is central to implementing a potential resource allocation plan.

The epidemiology research community has invested resources in order to understand and predict disease distribution within a premise as well as between premises by leveraging statistical epidemiological models [8]. These models are often represented as stochastic discrete events, which demand hundreds to thousands of input parameters and turn to be compute-intensive. The North American Animal Disease Spread Model (NAADSM) is one such model. This model was developed after the 2001 UK outbreak as the refinement of previously used models for policy making in the case of an FMD outbreak [9]. It has been scrutinized by over 300 epidemiologists and veterinarians, and it is used by United States Department of Agriculture (USDA) to plan for disease incursions [9]. NAADSM is being used to model various infectious FMDs, highly pathogenic avian influenza, swine flu, and pseudorabies [10], [11], [12]. The models authenticity and origins make it a perfect fit for this study.

A single commodity machine completes a typical NAADSM simulation run in nearly 70,000 seconds [13], which proves the necessity of a processing platform that is perfectly customized for the influential analysis. In NAADSM, disease biology parameters include transmission via airborne or direct contact; control measures (such as vaccinations); and effectiveness of vaccines, quarantines, shipments, and veterinarian visits. Since the simulation is stochastic, each set of input parameters is executed several times to gain statistical confidence in the results. These iterations contribute

to the overall representation of the output variables' probability distributions. Key outputs used during planning include the disease duration, number of infected animals, and depletion of vaccine stockpiles. While this study targets livestock disease outbreaks, the methodology described here is broadly applicable to systems where entities are organized into large networks and the spread of information, such as pathogens, ideas, or tracked movements, is based on relationships between entities.

1.1. Scientific Challenges

This section introduces the set of scientific challenges that arise in the timely determination of premises' influence from voluminous simulated epidemiology data.

- **Dataset Size:** As mentioned earlier, NAADSM simulates each input case several times, each of which produces numerous output parameters that are stored in a single file, which must be processed in order to extract relevant information to perform the influential analysis. As input parameters can be in the order of thousands, the dataset size and computation demands outgrow the capability of single commodity computer.
- **Timeliness:** The data analysis must be completed earliest on the computing platform in order to get timely results. Given the enormous data and related inherent challenges, such as disk I/O and the capability of single commodity machine, the likelihood of electing any existing solution that fulfills requirements in their entirety is non-existent.
- **Scalability:** The proposed approach should be scalable enough to retain its purpose in case of variance in the nature of the data in terms of number of premises and interconnectivity between them, as well as computing resources. This ensures the validity and uniformity of proposed approach.
- **Accuracy and Interoperability:** The analysis must be reasonably accurate, and it must support interpretability by explaining why a premise is considered highly influential. This is critical for fine-tuning outbreak responses.

1.2. Research Questions

The following research questions are addressed with this work:

(1) **What data structure(s) allow the representation of disease spread interactions for analysis?** Specifically, the proposed approach must be able to capture infection information from the simulation output while preserving the cumulative dynamics of disease spread.

(2) **How is the influence of each premise measured?** This involves aggregating data from millions of simulation runs as well as the selection of an appropriate approach, which should be able to justify influence measures of premises.

(3) **How is the analysis achieved at scale in the least time?** Given the data volumes involved, repeated sweeps over on-disk data must be avoided, and the analysis should be executed concurrently on multiple machines, which guarantees the least time provision. In order to achieve scalability, the analytics platform should be implemented by enhancing a state-of-the-art distributed computing solution.

1.3. Overview of the Approach

This section gives an overview of the analytics approach used in this work, which is designed to overcome the challenges presented and to reasonably answer the research questions. The proposed approach for determining the influence measure of premises from the voluminous simulated epidemiology dataset proceeds in the following manner:

(1) The information related to infection interaction between premises, which is essential for the analysis, is extracted from NAADSM simulation outputs.

(2) The Disease Transmission Network (DTN), later used by the analytics algorithm, is constructed by aggregating the information extracted in the previous step.

(3) The PageRank algorithm [14] is applied to the DTN in order to find relative influence measure of premises in NAADSM and to compare the results with the super-spreader analysis [15].

The above steps briefly describe the operations involved in the influential analysis. In order to perform these in an efficient and timely manner, the computation platform must be optimized. The following points briefly explain the way existing distributed data storage and processing technologies are leveraged and customized to build such a platform.

- The data extraction was accomplished using the customized input handler with the Apache Hadoop Distributed File System (HDFS) [16], a reliable and fault-tolerant distributed file system that is the storage component of the open source distributed storage and processing framework, Apache Hadoop.
- The creation of the DTN as well as the execution of the PageRank algorithm was performed on the customized version of the in-memory distributed graph processing framework, Apache Spark GraphX [17]. This version provides noticeable enhancements over the existing one.

The epidemiology dataset, which comprises of simulation outputs from NAADSM, encompasses multiple representative scenarios and iterations, which were processed to extract and record millions of infection incidents. This included tracking the number, source, destination, and duration of infections. This information was encoded in the DTN, which is a weighted, directed graph that summarizes the infections between premises. Nodes in the DTN are premises, and edges represent infection transmissions. The direction of traversals in the DTN varies depending on the algorithm underlying the analysis.

Once generated, the DTN was used to determine influence measure of premises. In the analytics workflow, the PageRank algorithm, originally used in the Google search engine to estimate the importance of web pages [14], was leveraged to determine these influence measures. The PageRank value for each premise in the DTN was calculated; if a premise had a higher PageRank value, we considered the premise to be more influential in the disease outbreak.

Apache Spark GraphX is an in-memory distributed graph processing framework [14]. PageRank being a graph algorithm, GraphX is a well-suited execution environment for it. Ideally, graphs

are partitioned and input to the algorithm. This partitioning causes communication and load balancing overheads. During this work, the performance of GraphX was enhanced by improving the underlying 2D graph partitioning scheme such that every vertex in the graph saves at most two network transfers while distributing nearly an equal amount of load among partitions, ultimately allowing the algorithm to complete faster.

1.4. Thesis Contribution

This thesis presents the approach for determining influence of premises from simulated voluminous epidemiology data as well as the enhanced version of a cutting-edge distributed graph processing framework. Specific contributions of this work include:

- Design of a graph-based data structure, the DTN, which preserves the cumulative dynamics of disease spread across space and time. This data structure supports traversals that are needed for analysis.
- Determination of premises' influence by harnessing and adapting the PageRank algorithm in the context of epidemiology.
- Customized and efficient platform design that avoids repeated I/O passes over the dataset and compactly encodes results in the memory-resident DTN along with the enhanced graph partitions, which reduces the computation time of determining relative influential measures.

1.5. Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 provides a detailed background encapsulating NAADSM and graph partitioning, as both topics play a pivotal role in comprehending this work. Chapter 3 gives an overview of previous methodology related to this work in the context of analytics and graph partitioning. Chapter 4 describes the methodology to determine relative influence measure of premises, which includes data extraction, creation of the DTN, and

application of the PageRank algorithm. Chapter 5 contains a discussion of the customization of the execution environment, mostly concentrating on the enhancement of the existing 2D graph partitioning scheme in Apache Spark GraphX. Chapter 6 is a presentation of the evaluation of the proposed approaches for the determination of influence measure as well as for the graph partitioning. Chapter 7 concludes this work with answers to the research questions and a discussion of future avenues concerning the expansion and refinement of the current methodology.

CHAPTER 2

Background

This chapter details the background knowledge required to understand the work performed in this thesis. An introduction to simulation in NAADSM is critical because analytics is accomplished on outputs generated by it. The history and fundamentals of graph partitioning are presented to facilitate a clear understanding of the methodology used; the customized computation platform is established in this study by enhancing one of the graph partitioning schemes provided in Apache Spark GraphX.

2.1. NAADSM

The North American Animal Disease Spread Model (NAADSM) is a stochastic simulation of highly contagious disease outbreaks in animals that facilitates strategy development and decision making [9]. In this model, groups of livestock, called *units*, are the basis of the simulation. Note that the terms, *premise* and *herd*, are also being used to refer to a group of animals. Disease spread between units is influenced by production types, intergroup similarities (shipment rates, infection rates, etc.), relative locations, and distances between herds. When a unit is infected, it follows a natural cycle of disease states consisting of susceptible, latent, subclinically infectious, clinically infectious, naturally immune, vaccine immune, and destroyed. This cycle can be interrupted by disease control strategies including quarantine, destruction, and vaccination. The disease spread among the units can happen by any of the following three methods: direct contact, indirect contact, and airborne spread. Stochastic processes drive all operations in the model, and they are based on user-defined distributions and relational functions. NAADSM input parameters can be of six types: yes/no values, integers, floating point numbers, probabilities, probability density functions, and relational functions. Collectively, these parameters form a scenario, and each scenario represents a simulated disease outbreak. Because the simulation is stochastic, it is run for several iterations (32

per scenario, in this study) to gain confidence in the output distributions. Each of the scenarios as well as iterations is completely independent of each other. To reduce the overall execution time of the simulation, NAADSM can be parallelized over a cluster of computing resources in a fault-tolerant fashion [18].

2.1.1. ***Dataset:*** The subject dataset was derived from a sensitivity analysis that explored the NAADSM parameter space to produce multiple valid combinations of inputs set in Colorado, USA [19], [20]. This process generated 100,000 scenario variants that were executed 32 times for a total of 3.2 million outputs (6.26 TB). In this particular scenario, a single initial herd is infected, with disease spread eventually encompassing tens of thousands of premises. The output of the simulation contains attributes representing the disease status of individual premises and how the infection spreads across premises within the network. These outputs also account for topological characteristics such as connectivity between the premises, proximity, and contact due to movements.

NAADSM output is extremely data-intensive even for a single simulation run. Scenarios are in the order of hundreds of thousands, each of which is executed for 32 iterations. Furthermore, an output file is created for each of the iterations; hence, the file count of a single simulation run is in the order of millions. These files contain outbreak-related information for each of the simulation days, meaning, if a scenario iteration makes a simulated outbreak last for n simulated days, the corresponding output file consists of data for that number of simulated days. Moreover, this model outputs the disease state for each of the herds, indicating infection propagation along with summary statistics that include information related to categorical control strategies for each simulation day.

2.2. Graph Partitioning

Lately, graphs and related algorithms have found applications in a wide variety of domains including social networks, recommendation systems, web documents, routing, and many more. Graphs represented in these applications have continued to emerge at an astonishing rate, and

trends are clearly directing the growth. The Facebook friends graph has more than 1 billion vertices (users) and nearly 1 trillion edges (friendships) [21]. Moreover, many machine learning techniques can be represented as graphs, such as Deep Neural Networks. Applications in these domains demand nearly real-time results. Although the computing and storage capacity of one machine has increased exponentially, a single machine is unable to satisfy real-time requirements completely. This leads to the advent of various distributed graph computation systems, also called *distributed graph-parallel systems*, which operate above the cluster of commodity hardware [22], [23], [24], [17]. These systems have employed the *"Think Like a Vertex"* [25] programming model and exposed developer-friendly APIs wherein any graph algorithm can be expressed from the perspective of a vertex. Ideally, input graphs are partitioned and stored on machines within a cluster to execute graph algorithms quickly and use cluster resources completely. Evidently, graph partitioning is the primary factor in deciding the performance of a graph algorithm in a distributed environment. Graph partitioning raises following concerns:

- **Ingression time:** This refers to the time that is taken by the system to partition and load the graph before starting the actual execution of the algorithm. Clearly, it completely relies on the partitioning strategy. Some strategies take more time to generate the partitions than others.
- **Communication cost:** This refers to the amount of network transfer between partitions while executing the algorithm. Ideally, partitions are stored on different machines. Therefore, some amount of communication is required between these machines in order to achieve the necessary synchronization. Obviously, this plays a prominent role in the performance of the partitioning scheme, and accordingly, the graph algorithm.
- **Load balancing:** This refers to the quality of load distribution, which ultimately boils down to the number of vertices and edges assigned to each partition. Clearly, improper distribution leads to under- or over-utilization of cluster resources.

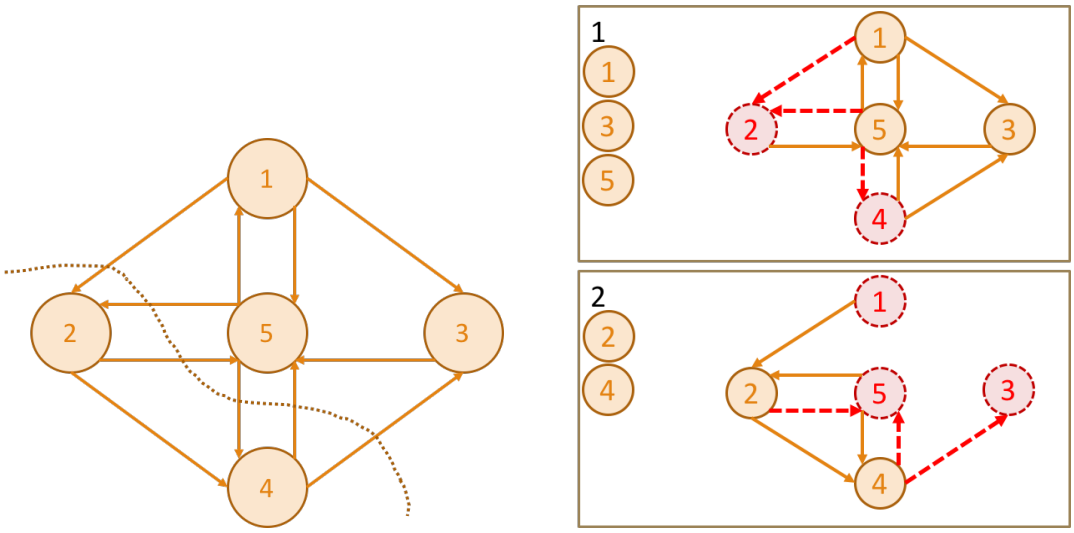
Balanced graph partitioning is classified as an NP-complete problem [26]. This means that the solution to a problem can be acquired by heuristics or approximation algorithms. With the

arrival of graph-parallel systems, the research community has started taking interest in streaming graph partitioning algorithms, which partition a vertex or an edge based only on the information that belongs to themselves, without having knowledge about the complete graph. Primarily, these algorithms are divided into the following two categories:

(1) **Edge-cut:** The motivation behind the algorithms in this category is that vertices are equally distributed among partitions and edges are cut and distributed across partitions in order to produce a complete subgraph, hence the name *edge-cut*. It is worth noting that, cut edges along with the corresponding vertices are replicated and passed according to the requirement between partitions. Due to this characteristic, the communication cost associated with edge-cut algorithms is directly proportional to the number of edges cut. Clearly, each partition will be responsible for an approximately equal number of vertices, whereas that of edges will differ, which determines the load-balancing factor.

Figure 2.1 shows an example of edge-cut partitioning. A graph with 5 vertices and 11 edges is given in Figure 2.1a. The goal is to divide this graph into 2 parts, such that each of the parts is a complete subgraph. As mentioned earlier, vertices are partitioned first. In this example, they are partitioned using the elementary hash partitioner, which is $v\%n$, where v is a vertex identifier and n is the total number of partitions. Hence, vertices 1, 3, and 5 are placed into partition 1, and vertices 2 and 4 are placed into partition 2. This partitioning makes all of the edges having vertices from both of the sets miserable. Those edges are demonstrated using a hypothetical cut in Figure 2.1a. Hence, edges (1,2), (5,2), (5,4), (2,5), (4,5), and (4,3) are ambiguous regarding their partition, but the rest of the edges are placed into the same partition where both of their vertices are present. In order to place those miserable edges, a deterministic scheme has to be decided. Suppose, those edges would be placed into their destination vertex's partition, meaning edge (1,2) would be placed in the same partition where vertex 2 is placed. After this, the subgraphs are incomplete. For example, all of the edges of vertex 2 are not in the same partition as it is; edge (2, 5) is in partition 1. In order to function, these subgraphs

need to be completed. Therefore, some of the vertices and edges have to replicate themselves into other partitions. The vertices and edges denoted by red dashed lines in Figure 2.1 are those with such properties, and they are the cause of communication between the partitions.



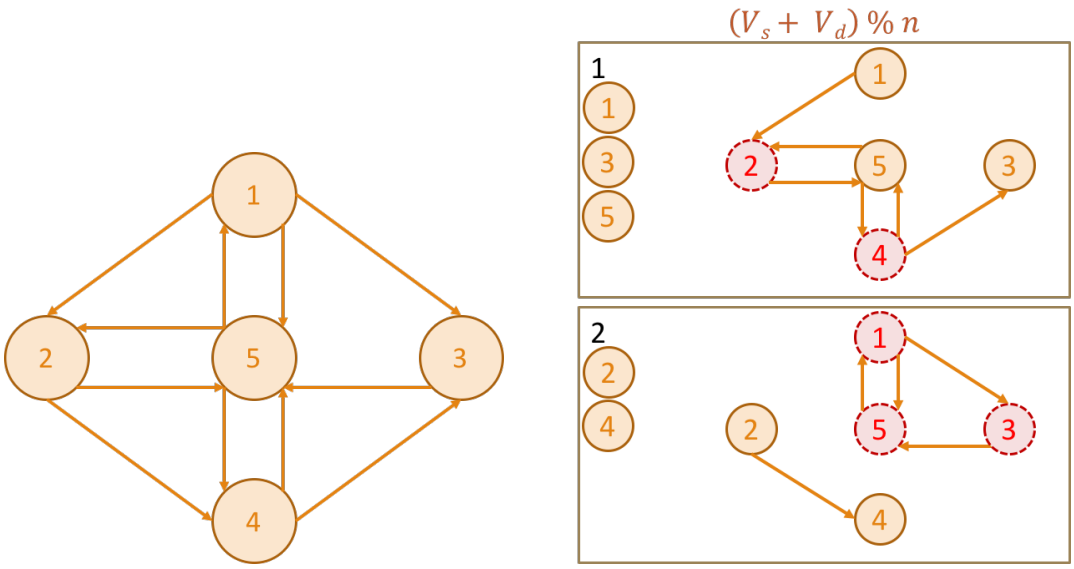
(a) Actual Graph with Cut Edges *(b) Partitioned Graph*

Figure 2.1. Edge-cut Partitioning

(2) **Vertex-cut:** The motivation and fundamentals behind the algorithms in this category are completely opposite to that of edge-cut algorithms. Unlike edge-cut, edges are equally distributed among the partitions, and vertices are cut and replicated across the partitions in order to produce complete subgraphs, hence the name *vertex-cut*. As opposed to edge-cut, vertex data is passed between the partitions, so the communication cost is directly proportional to the number of the vertex replicas, and the load balancing factor is determined by the number of edges assigned to each of the partitions.

Figure 2.2 shows the vertex-cut partitioning. As in the previous example, a directed graph with 5 vertices and 11 edges is provided. The goal is also the same, but the technique is different. Unlike the edge-cut process, there is an attempt to distribute the edges equally among the partitions. Suppose, this has been achieved using the formula shown on top of Figure 2.2b, which is $(V_s + V_d)\%n$, where V_s is the source vertex identifier, and V_d is the destination vertex identifier. Because of this, edges (1,2), (5,2), (2,5), (5,4), (4,5), (4,3) are placed into

partition 1, and the rest of the edges are placed into partition 2. Furthermore, suppose vertices are partitioned using the elementary hash partitioner, $v\%n$. Like the previous example, this partitioning also creates subgraphs that are incomplete. However, as opposed to the previous scheme, where both edge data and vertex data are passed between partitions in order to make them complete, this strategy just passes vertex data. This is represented by red lined vertex replicas in Figure 2.2b, which are the only source of network communication in this case.



(a) Actual Graph *(b) Partitioned Graph*

Figure 2.2. Vertex-cut Partitioning

The following measures are primary in evaluating the quality of partitions.

- **Replication Factor:** This measure quantifies the amount of replication of graph-property. With the edge-cut strategy, the property under investigation is the set of replicated edges. In the case of the vertex-cut strategy, the average vertex replica is investigated, which is nothing but the ratio of the total number of vertex replicas to the total number of vertices. Clearly, this measure is directly proportional to the communication between partitions.
- **Load Balancing Factor:** This measure shows the distribution of load among partitions. This factor is represented by the standard deviation of the number of edges for which each partition is

responsible. A lower standard deviation indicates better utilization of cluster resources and vice versa.

The quality of any partitioning scheme totally relies on the measures mentioned above. An ideal scheme incurs minimum replication. Thereby such a scheme minimizes communication, and it distributes a nearly equal number of vertices and edges among partitions. In general, the ingression time is directly proportional to the quality of partitions, meaning that a partitioning strategy with a higher ingression time may produces better quality partitions and vice versa.

Early distributed graph-parallel systems, Pregel [22] and GraphLab [24], employed the edge-cut partitioning scheme. The research on these systems discovered serious load balancing issues for edge-cut partitioning in the case of Power-law graphs [27], the graphs that follow Power-law degree distribution and are alternatively called real-world graphs or natural graphs, due to massively imbalanced edge distribution. Therefore, PowerGraph [27] employed the vertex-cut partitioning scheme and reported significant improvement in the quality of partitions for Power-law graphs. Following this, later distributed graph-parallel systems, including Apache Spark GraphX [17], have also employed vertex-cut partitioning along with various implementations of it. Edge-cut and vertex-cut partitioning schemes serve different purposes. Edge-cut is well-suited for graphs with a high number of low-degree vertices since there exists a high possibility of assignment of all edges of a vertex to the same partition. In contrast, vertex-cut is best fit for graphs with a small number of high-degree vertices, like the Power-law graph, since edges of these vertices are evenly distributed and, consequently, balanced partitions are generated.

During this work, the 2D graph partitioning scheme, an implementation of vertex-cut partitioning provided by Apache Spark GraphX, has been studied thoroughly and enhanced for the analytics platform, which is explained thoroughly in Chapter 5.

CHAPTER 3

Related Work

This chapter is an overview of previous work in this area of research that largely explains the absence of essential properties of the ideal approach. Recent findings compel researchers to improve upon existing methods, justifying the necessity of this work.

During an outbreak, influential herds transmit the disease to their neighbors, ultimately making outbreaks last longer or become more severe. As a result, the influence of a herd depends largely on the influence of its neighbors. The analysis of influence in epidemiology has been a considerable area of study, with much of the work revolving around the various characteristics of infected entities and their influence on disease transmission [28], [29]. However, these approaches generally examine standalone characteristics and not the underlying network or relationships that underlie disease spread.

Social Network Analysis (SNA) focuses on human interactions in social networks, but it can be applied to analyzing animal epidemics as well. Considerable research has been conducted on influence in social networks [30], [31], [32], [33], [34]. The Independent Cascade (IC) model and Linear Threshold (LT) model are commonly used to describe the influence of nodes in directed graphs. The LT model declares a node as either active or inactive based on a threshold and the sum of weights of neighboring edges. On the other hand, in the IC model, each active node is given an opportunity to activate its inactive neighbors, with the process repeating until a steady state is reached [35], [36]. In this case, active nodes are considered to be highly influential. However, since both of these methods rely on binary states (active or inactive), relative measures between nodes are not supported.

Cha et al. studied the influence of users on Twitter based on three metrics: in-degree, retweets, and mentions [34]. This approach uses Spearman's rank correlation coefficient to compare user influence, and it evaluates the behavior of the three metrics for highly influential users. An approach

outlined by Khrabrov and Cybenko uses daily mentions of users on Twitter as a basis for calculating different rank metrics such as PageRank, dRank, and StarRank to determine the influence of users [37].

Aggarwal et al. proposed two algorithms, *SteadyStateSpread* and *RankedReplace*, to determine information flow representatives, a small group of authoritative figures to whom the release of information leads to maximum spread [31]. *SteadyStateSpread* iteratively finds a candidate set of nodes with higher steady state flow values as candidate representatives. This method ignores the structural relationship of nodes, which inspired the *RankedReplace* algorithm. In *RankedReplace*, nodes are replaced iteratively and sorted in descending order by their steady state flow values to maximize total flow [31]. Moreover, Budgaga et al. [19] proposed a framework that explores statistical ensemble and learning methods for predictive analytics. This framework uses dimensionality reduction and ensemble techniques to improve the performance and accuracy of forecasting models generated in a distributed environment.

Lately, a variety of methodologies and query-based frameworks have been introduced to perform anomaly detection over a distributed environment. Malensek et al. proposed a framework which supports ad hoc queries that allow users to trade off accuracy versus timeliness [38]. Moreover, there is support for programming analytic operations based on queries [39]. Optionally, the query can be spatially constrained [40]. Support is also available for detecting anomalies based on spatiotemporal characteristics [41].

Real-world graphs tend to be massive. They need to be partitioned and stored on a cluster of commodity machines to process graph algorithms quickly. Partitions cause load balancing problems, and synchronization between these partitions leads to communication-related issues. Optimal graph partitions reduce communication costs, concurrently providing equal distribution of workloads. Therefore, the associated research community has been concentrating on real-time graph partitioning schemes, partitioning without knowledge of the complete graph. Recently, this

approach has generally been used for streaming graphs. In addition to that, researchers have explored graph-partitioning for scheduling in the context of scientific workflows [42], [43], [44].

PowerGraph, a distributed graph processing framework, has introduced the Oblivious graph partitioning scheme [27]. This strategy demands information regarding previous edge assignments to perform a current assignment more precisely. However, it refrains from collecting information from other partitions and completely depends on local knowledge to avoid communication overheads. It's obvious that this is just a heuristic, although it performs better than random assignments. Sajjad et al. have proposed the *HoVerCut* methodology, which can be coupled with existing graph partitioning algorithms [45]. This method uses a distributed multithreaded environment where each thread is called a *subpartitioner* and runs the partition algorithm inside. Additionally, *HoVerCut* employs a windowing technique that produces a trivial shared state because it generates batch as opposed to immediate updates. This method is said to be the foundation of optimal communication between partitions. Instead of a partitioning algorithm, *HoVerCut* can be considered to be an addendum. PowerGraph has introduced the *PDS* scheme, which leverages Perfect Difference Set [46]. This method requires $(p^2 + p + 1)$ partitions, with p being a prime number. Due to this hard constraint, this scheme is rarely used in real-world applications.

There have been a considerable number of graph partitioning algorithms that place emphasis on the degree of the vertex to employ better heuristics [47], [48], [49]. These algorithms assign vertices to partitions based on a random hash function and attempt to discover a finer technique for edge partitions. Xie et al. have suggested a degree-based hashing algorithm, *Libra* [48], based on the idea that better partitions are achieved if more vertices with higher degrees are cut. Therefore, having two choices for each edge, the source and the destination vertex partition, this algorithm assigns an edge to the vertex partition with a lower degree. Petroni et al. have proposed a stream-based, vertex-cut partitioning algorithm, *High-Degree Replicated First (HDRF)* [47]. This algorithm attempts to replicate high-degree vertices and maintains locality for low-degree vertices. Chen et al. have employed an algorithm, the *Hybrid-cut* [49], which differentiates partitioning

schemes for low-degree and high-degree vertices based on the in-degree of a vertex; however, this method is applicable only to directed graphs. It assigns all edges to their respective target vertex partitions followed by reassignment of edges with high-degree vertices, identified by degree threshold, to a respective source vertex partition. These algorithms claim to produce better partitions for power-law degree graphs in terms of communication and load balance since they share a similar foundation, which is to replicate high-degree vertices. Because deciding the partition for an edge with the *Hybrid-cut* is based on the degree of vertices, vertices and their degrees have to be coupled, before the algorithm functions. This constraint burdens the graph ingression time because coupling requires communication between partitions. *Granules* and *Neptune* provide support for low latency processing and analytics datastreams [50], [51]. In addition to these features, *Granules* leverages the NaradaBrokering framework in order to disseminate data streams in cluster and grid settings [52], [53].

CHAPTER 4

Determining Influence

This chapter describes the analytical approach that was taken in this research to accomplish the goal of finding influence measures. This description includes justification of the approach, creation of the Disease Transmission Network (DTN), preliminary geospatial analysis, the PageRank algorithm, and the application of the PageRank algorithm to the DTN.

The actual problem discussed in Chapter 1 was that of finding relative influential measures such that a limited amount of available resources can be used effectively in the case of a disease outbreak. Most approaches discussed in Chapter 3 facilitate binary outputs for an entity's being influential or not. Hence, they are unable to solve the problem in its entirety. A type of methodology that considers the interaction between entities and finds the relative influence of each of them is the best-fit solution. PageRank is such an algorithm that is in the process of development with very convincing results. Google search results are powered by this algorithm. Empirically, it is very unlikely not to find a desired article in the first 15 to 20 Google search results. These articles are sorted based on the descending order of their PageRank values. Furthermore, these values are related to the search term (purpose) meaning; PageRank values of an entity differ based on the purpose. These attributes of the PageRank algorithm are completely aligned with the preferable solution, which makes it the best-fit method for considering interactions between entities to determine influence. The remainder of this chapter describes the steps required to apply the PageRank algorithm to the subject dataset, which includes building the DTN and calculating the PageRank values of the herds under consideration.

4.1. Disease Transmission Network (DTN)

PageRank is a graph algorithm. In order for it to be applied, the subject dataset, the NAADSM simulation output, had to be converted to a sophisticated network that is able to represent disease

interactions between the herds. This goal was achieved using the *Disease Transmission Network (DTN)*. The DTN is a weighted, directed graph of infection interactions where vertices represent herds and directed edges represent the disease transmission between herds. Furthermore, edge weights depict the influential relationship between the corresponding herds. Since simulation output is data-intensive, a cluster of commodity hardware was leveraged to perform the analysis in a concurrent fashion. Generation of the DTN from simulated output is explained in the steps outlined below. The complete workflow of analytics tasks performed is displayed in Figure 4.1, which includes generation of the DTN from the subject dataset, the inverted graph, and the application of PageRank.

(1) NAADSM creates a compressed file per scenario iteration, which is decompressed and staged to a reliable and fault-tolerant distributed filesystem, HDFS. HDFS partitions these files into 64MB or smaller chunks, and it stores a predefined number of replicas in the Hadoop cluster.

(2) Since content in the output files follows a specific format, the infection propagation pattern is extracted using custom Hadoop Inputformat. File chunks, alternatively called input splits, are scanned for the infection propagation pattern per scenario iteration, and a directed graph is formed by ordinating the extracted patterns, which depicts disease spread during a simulated outbreak. Since the initially infected herd, alternatively called the source herd, is the same for each scenario iteration, this herd has been removed from this graph to preserve the stochastic behavior. Nodes in this graph are herds, and directed edges represent infection propagation.

(3) Millions of such graphs are aggregated in order to build a sophisticated network, which has been done in the following manner:

   (a) For each of the edges in the graphs generated during the previous step, an edge is placed between the respective vertices in the resultant graph with a unit weight. As an outcome, the resultant graph is directed, where each of the edges represents an infection interaction from any of the scenario iterations. The topological interpretation of this fact is that

multiple instances of the same edge between any possible pair of vertices with a unit weight may exist, which can be observed in Figure 4.1.

(b) Identical edges, those with same source and destination, are aggregated by summing their weights. After this step, there is only one edge between any possible pair of source and destination vertices. Furthermore, the weight of any edge is an integer number, which represents the total number of infection interactions observed combining all scenario iterations for the simulation.

(c) These edge weights should be converted into a representation which strongly portrays infection interaction between herds. This is attained by transforming these integer weights to the percentage weights using the following formula.

$$weight(A, B) = \frac{\text{Number of times } A \text{ infects } B}{\text{Number of times } B \text{ is infected}} \tag{1}$$

where

$weight(A, B)$ = Weight of the edge from A to B in the DTN which represents the rate at which A infects B;

Number of times A infects B = Weight of the edge from A to B in the graph generated after step (b); and

Number of times B is infected = Sum of weights of incoming edges to B in the graph generated after step (b).

To understand the above formula, consider the following example. Suppose herd B is infected 5 times by herd A, 1 time by herd C, and 4 times by herd D. The edge from A to B in the DTN will have a weight of $\frac{5}{10} = 0.5$, where the numerator, which is 5, is the number of infections from A to B, and denominator, which is 10, is the sum of weights of all incoming edges to B.

The DTN created by following these steps precisely encodes the needed information regarding infection interactions and the outbreak. Using the DTN, the subject dataset, which is comprised of 6.26 TB of data, is concisely expressed by just over 132 MB. Following are two noteworthy properties:

- The weight of an edge depicts the importance of the source in infecting the destination, focusing on the total number of times the destination is infected (denominator), regardless of the number of infections that originated from the source.
- For every vertex in the DTN, the sum of the weights of all incoming edges is one, but nothing can be observed about the properties of the outgoing edges.

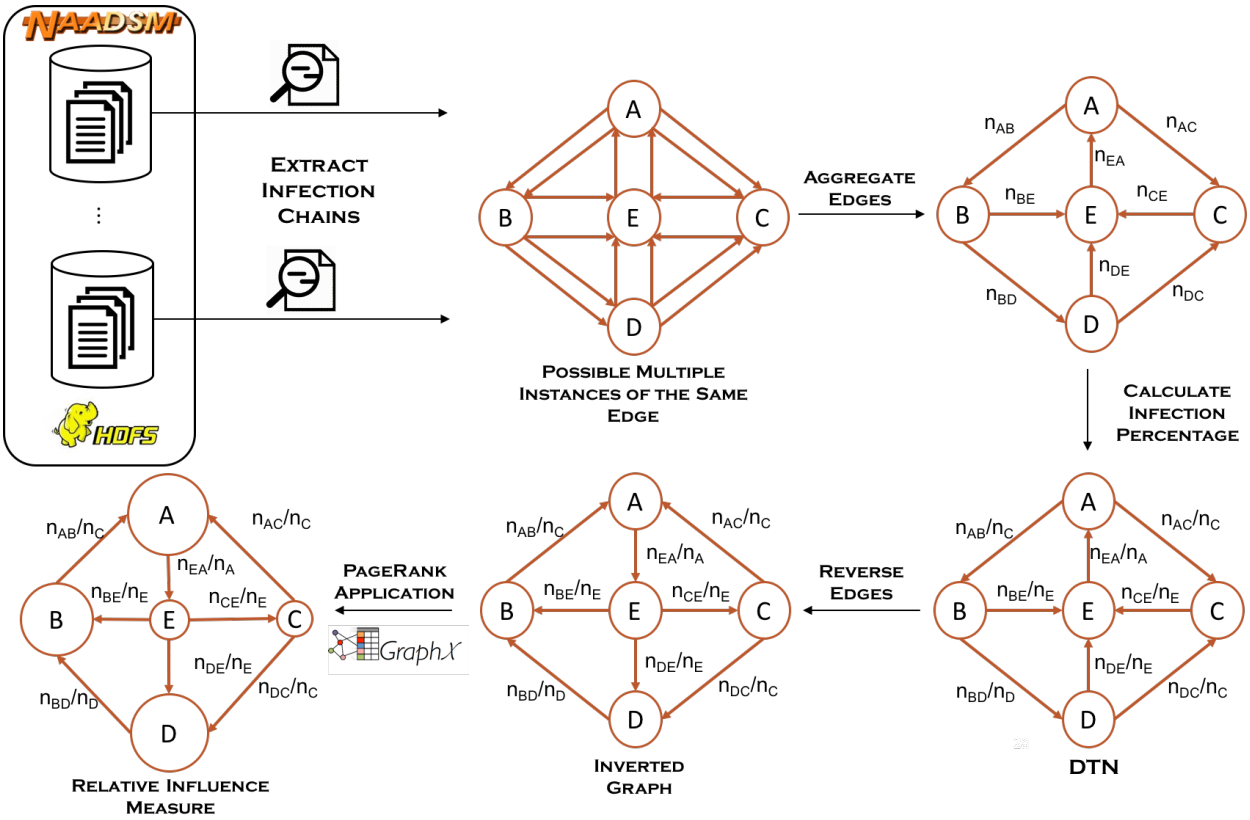


Figure 4.1. Analysis Workflow

4.2. Preliminary Analysis: Geospatial Distance

After the initial creation of the DTN, the correlation analysis on the geospatial distance between the units and the rate at which a unit infects others was performed. This evaluation served to test the functionality of the DTN and gain insight as to how disease spread interactions behave spatially. Using the DTN, infection rate between the herds was calculated using the following formula:

$$\text{InfectionRate}(A, B) = \frac{\text{CountOfInfections}(A, B)}{\text{SourceOfInfection}(A)} \tag{2}$$

where

SourceOfInfection(A) = total infections from unit A, and

CountOfInfections(A, B) = total infections that unit A transmitted to unit B.

The infection rate, as defined in the Formula 2, as well as the geospatial distance between the herds is calculated for every pair of herds in the DTN. Using these points of comparison, the Pearson Correlation Coefficient (PCC) calculated for this data was -0.048, indicating that there is almost no correlation between the infection rate and distance between the herds. This experiment demonstrates that within a particular scenario, a diseased unit is unlikely to infect a herd in close proximity significantly more than those at greater spatial distances.

4.3. PageRank Algorithm

PageRank was proposed by Page et al. [14] and used by the Google search engine to sort search results by their relevance or importance. The algorithm assigns a PageRank value to each web page that describes the probability that a random surfer (randomly clicking on links) will arrive at the web page. The higher the PageRank value, the more important the web page is. In general, highly linked pages are more important than pages with a low number of incoming links. Furthermore, the PageRank value of a particular page determines how influential its outgoing links will be. If a page has very few input links but some of them are from highly linked web pages, the page is ranked

higher than a page that has more, but less important, input links. This means that a website can achieve a high PageRank value either by having a large number of incoming links or by being linked to from an important page. When applied to the DTN, a PageRank value represents the herd's capability of infecting others in a random disease chain, which is the influence of that herd on the entire network. Considerable research has been conducted on using PageRank to determine influence [30], [31].

4.4. Application of PageRank to the DTN

Construction of the DTN produces a weighted, directed graph, where the weight of each edge is the rate at which one unit is infected by the other. As a result, the sum of the weights of all input links must be equal to one. When a disease is transmitted from vertex A to vertex B, the interaction was modeled as *A is influencing B*. Similarly, vertex A influences all of its downstream neighbors. However, the PageRank algorithm computes the importance of entities based on the input links, but in this case the influence of a vertex is decided by the output links. Therefore, the direction of the edges is reversed in the graph without changing their weights to generate an inverted graph. This preserves the semantics of the network and allows usage of the PageRank algorithm without modification. Afterwards, the PageRank algorithm is applied with 25 iterations, and PageRank values are noted as the influence measure of corresponding herd.

This analytics approach is evaluated and compared with the super-spreader analysis [15]. Chapter 6 demonstrates the results of the comparison along with that of the scalability.

CHAPTER 5

Enhanced 2D Graph Partitioning Scheme

This chapter explains the 2D graph partitioning scheme implemented in the Apache Spark GraphX in detail. Moreover, the enhancement achieved in that scheme in order to build a customized version of GraphX is described thoroughly.

5.1. Existing 2D Graph Partitioning

As discussed in Chapter 2, modern distributed graph-parallel systems have availed the vertex-cut partitioning technique and provided various implementations of it based on heuristics. 2D partitioning, alternatively called grid-based partitioning, is one of these techniques. This scheme is an implementation of the grid-based constrained random vertex-cut strategy proposed by Nilesh et al. [54], which claims an upper bound of $2\sqrt{n} - 1$ on the vertex replication factor, where n is the number of partitions. A detailed explanation of this bound is provided later in this chapter.

The grid is a two-dimensional geometry. Graphs possess two properties, which are vertices and edges. The identifier space of vertices is one-dimensional, meaning that a vertex can be uniquely identified just by its vertex-identifier. However, the identifier space of edges is two-dimensional because source and destination vertex identifiers are required to uniquely identify an edge. Due to this requirement, the grid has been leveraged to hold edge properties. Analogously, if an entity could be uniquely determined by three properties, a three-dimensional cubical geometry would be the best-fit solution to efficiently map instances of an entity to the partition space. In this strategy, figuratively, partitions are assumed to be a grid of rows and columns. The same vertex identifier space is mapped to these rows and columns, sources on one and destinations on the other, using hashing techniques. Obviously, the intersections of these rows and columns represent corresponding edges. Because of this form of mapping, the intersections between identifier spaces, which are the edges, can be accommodated elegantly in the grid, which is the partition space.

5.1.1. *Methodology:* This section describes the step-by-step implementation of the 2D graph partitioning scheme, named *EdgePartition2D* [55], in Apache Spark GraphX. *EdgePartition2D* defines only the partitioning strategy used for edges because that of the vertices is predefined. Consider that there exists a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Every vertex in V has a vertex identifier of type long and a vertex property, while every edge in E has source and destination vertex identifiers of type long and an edge property. The goal is to divide this graph into n parts such that the partitions should incur minimum communication and distribution should be as balanced as possible. Following is the guide that explains the existing 2D graph partitioning strategy used in Apache Spark GraphX. The complete process is pictured in Figure 5.1.

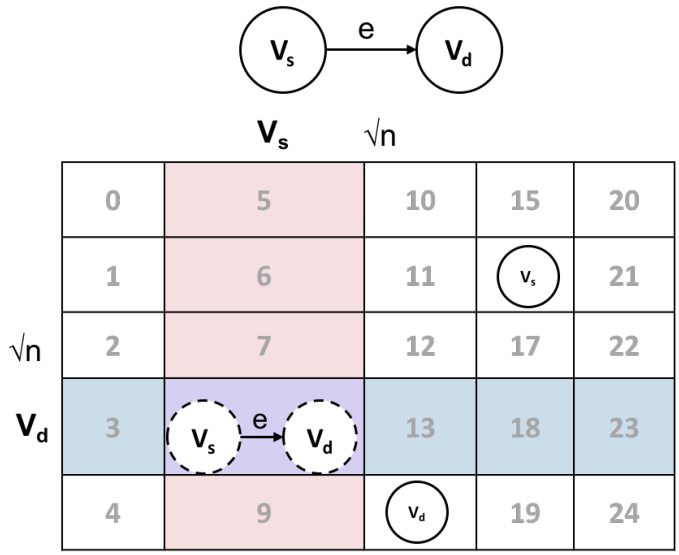


Figure 5.1. Vertices and Edge Placement

(1) Logically, partition space is a two-dimensional grid. If n is a perfect square, then the grid will have equal numbers of rows, hereafter called *rows*, and columns, hereafter called *cols*, that is $\sqrt{n}$. On the other hand, if it's not square, the grid could have differences in *cols* and *rows*, where *cols* can be at most one greater than *rows*.

- In the case of n's not being a perfect square number, *cols* is the ceiling part of the decimal value of $\sqrt{n}$. Furthermore, *rows* is expressed by the floor value of $(n + cols - 1)/cols$, which

also shows that *cols* can be at most one greater than *rows*. Every column has *rows* rows except for the last column, where the number of rows may vary from 1 to *rows*, hereafter denoted by $lastColRows$. For example, if $n = 27$, then $cols = 6$ and $rows = 5$, where the last column would have 3 rows.

(2) Vertices are assigned to partitions based on elementary modular hashing. For instance, a vertex with v as a vertex-identifier is assigned to the partition denoted by $v\%n$ in the case of n partitions. Consequently, vertices are equally distributed among partitions.

(3) Edges, on the other hand, are assigned by hashing both the source and the destination vertices.

- The source vertex identifier space is assumed to be mapped on the columns. Therefore, the column selection for any edge is performed using the source vertex, hereafter denoted by *src*. If n is a perfect square, *col*, is identified using $((src * mixingPrime)\%\sqrt{n})$. Otherwise, $((src * mixingPrime)\%n/rows)$, where $mixingPrime$ is a large prime number that is used to improve the balance of edge distributions [55].
- Unlike source vertices, destination vertices are assumed to be mapped on the rows. Hence, the row selection for an edge is accomplished by the destination vertex, hereafter denoted by *dst*. If n is a perfect square, *row* is identified using the $((dst * mixingPrime)\%\sqrt{n})$ function. If n is not square, *row* is identified using $((dst * mixingPrime)\%rows)$ if $col < cols - 1$ and $((dst * mixingPrime)\%lastColRows)$ otherwise [55].

(4) The edge property is assigned to the partition expressed by $(col * \sqrt{n} + row)$ in the case of n's being a perfect square, and $(col * rows + row)$ otherwise [55].

5.1.2. *Insights:* This section provides insights on the *EdgePartition2D* scheme from the perspective of the vertex replication factor (described in chapter 2) and justifies the upper bound claimed by the scheme. The section also includes an explanation regarding the load balancing that is achieved by this scheme, which helps to explain its popularity in the graph-parallel community for Power-law graphs.

- Looking at the *EdgePartition2D* scheme from the perspective of a vertex helps in perceiving the upper bound on the vertex replication factor. Consider a vertex with identifier v, where v could be a source as well as a destination in a graph. All the edges where v is the source vertex would be placed in the same column, *col*, of the grid, the partition space. Similarly, the edges where v is the destination vertex, would be placed in the same row, *row*, of the grid. Apparently, all the edges of v could either be in any of the $\sqrt{n}$ partitions of the col^{th} column or $\sqrt{n}$ partitions within the row^{th} row. Moreover, the intersecting partition of col^{th} column and row^{th} row is common among these $2\sqrt{n}$ partitions. Conclusively, any edge containing v has to be placed in any of the $\sqrt{n} + \sqrt{n} - 1 = 2\sqrt{n} - 1$ partitions, which justifies the upper bound on the vertex replication factor, which is clearly perceivable from Figure 5.2.

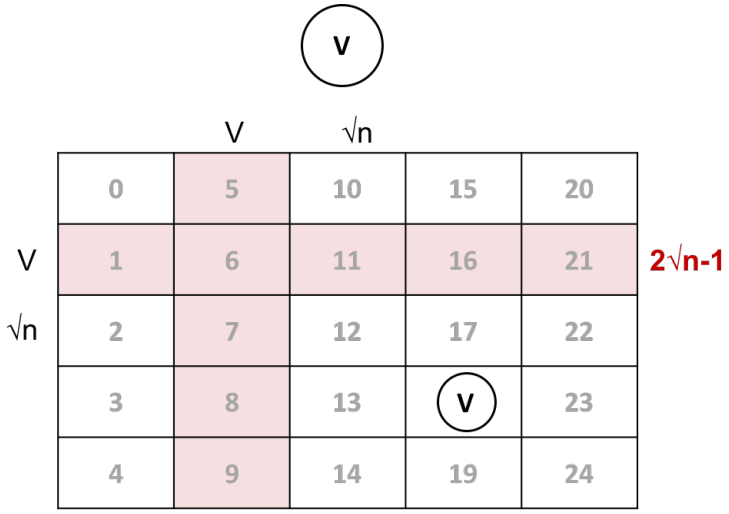


Figure 5.2. Upper Bound on the Replication Factor

- From the implementation of *EdgePartition2D* explained above, it is evident that same directional edges of any of the vertices are distributed among the partitions in the respective row or column, as opposed to the case in a few other partitioning strategies where all the edges with the same source or destination vertex would be placed in the same partition. The latter causes imbalanced load distribution for the high-degree vertices in Power-law graphs, which ultimately results in insufficient use of cluster resources.

5.2. Enhanced Version

EdgePartition2D is a foolproof implementation of the grid-based partitioning scheme. As explained earlier, it provides a perfect remedy for communication as well as load balance, two primary issues of any stream-based graph partitioning scheme. Vertices are evenly distributed, and edge distribution is nearly balanced; moreover, the vertex replication factor is under control. It is tough to beat these properties and think of an enhancement.

However, the vertex partitioning factor in *EdgePartition2D* is not completely optimized, so it has room for improvement. As mentioned in the earlier section, the partitioner is implemented using elementary modular hashing. Consider a grid of c columns and r rows. For a vertex v, all the edges where v is the source vertex are placed in partitions of the i^{th} column, $i <= c$, and all the edges where it is the destination vertex are placed in partitions of the j^{th} row, $j <= r$. Given this, the ideal place for v is the intersecting partition of the i^{th} column and j^{th} row. Instead, it is placed in a partition expressed by $v\%n$, where n is the total number of partitions.

The vertex partitioning scheme described above can be implemented by placing v in the partition where a hypothetical edge, with v as both the source and the destination vertex, would have been placed. Hence, the *EdgePartition2D* scheme can be leveraged to design such a vertex partitioner. In fact, it is fairly easy to implement such a partitioner by performing minor changes in the *EdgePartition2D* scheme. However, the implementation of this modified scheme raises load balancing issues. Consider that there are n partitions where n is a perfect square, so the grid would have $\sqrt{n}$ rows and $\sqrt{n}$ columns. As described in the earlier section, for a perfect square number of partitions, the column as well as the row for an edge are determined by the same hashing scheme, which is $(v * mixingPrime)\%\sqrt{n}$ in this case. Clearly, applying this vertex partitioner would result in all vertices being placed in the diagonal partitions of the grid. Any of the non-diagonal partitions would not be accommodating a single vertex. Apparently, this hashing is unable to distribute vertices among partitions equally. However, ideal load balancing can be achieved by employing

different hashing techniques for source and destination vertices, which is described in the following section.

- Given a perfect square number of partitions, n, the column for a vertex, v, can be elected using the $((v * mixingPrime)\%n)/\sqrt{n}$ function instead of $(v * mixingPrime)\%\sqrt{n}$. However, there is no change in row election, which was being performed using the $(v*mixingPrime)\%\sqrt{n}$ function. Since, the newer approach selects a column by both the modulo and the division operation, instead of just the modulo as in the previous approach, the column index would differ from the row index. This enables vertices to disperse over the grid. For example, $n = 100; v = 1546; mixingPrime = 1$. With the previous partitioner, v would be placed into $(1546 * 1)\%10 = 6^{th}$ column, and $(1546 * 1)\%10 = 6^{th}$ row in the grid of 10 * 10. With the newer approach, it would be placed into $((int)((1546 * 1)\%100/10)) = 4^{th}$ column and 6^{th} row.
- An approach similar to this is already followed for a non-perfect square number of partitions.

5.2.1. *Insights:* This section explains the insights of the new vertex partitioner described above in terms of replication factor and load balancing.

- Because the vertex is placed into the intersecting partition of the row and the column where the hypothetical edge having that vertex as the source and the destination would be placed, the vertex has to replicate itself to $\sqrt{n} - 1$ partitions in a column and $\sqrt{n} - 1$ partitions in a row. Hence, the upper bound on the replication factor would become $2\sqrt{n} - 2$, which is one less than the existing implementation, illustrated in Figure 5.3. For an iterative algorithm like the PageRank, this turns out to be two network transfers less per vertex per iteration. This reduction of the upper bound enhances the performance of the algorithm.
- Due to the difference in hashing techniques employed for source and destination vertices, their distribution throughout the grid would be balanced even in the case of a perfect square number of partitions.

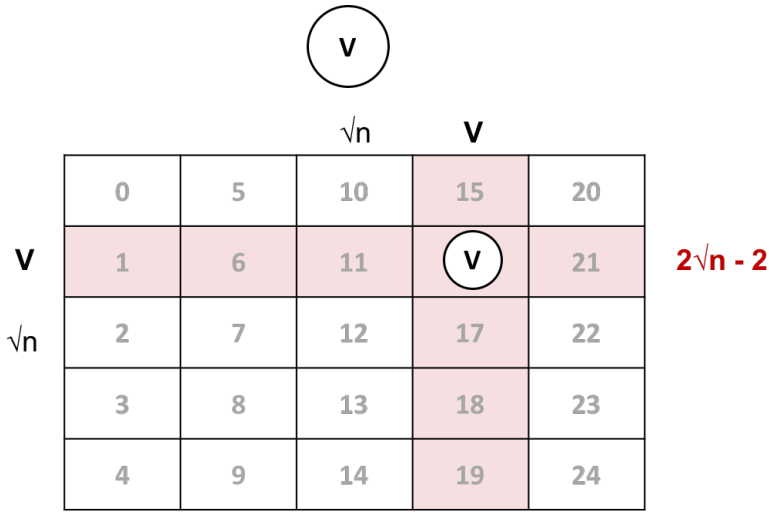


Figure 5.3. Enhanced Approach with Replication Factor

- Although in the modified partitioning scheme, source vertex hashing is different from that of *EdgePartition2D*, hashing of the destination vertex is unchanged. Therefore, the partitions identified by the *EdgePartition2D* scheme and modified partitioning would differ for a particular edge, but untouched destination vertex hashing and balanced vertex partitioning ensure that the edge distribution is balanced.

The results of a performance comparison of the enhanced version and the existing *EdgePartition2D* scheme are displayed in Chapter 6. This chapter includes results in terms of graph ingression time, replication factor, workload distribution, and execution time.

CHAPTER 6

Evaluation

In this Chapter, the evaluation of the proposed analytics approach is demonstrated, which includes a comparison of influence measure to the super-spreading analysis with an additional analysis of seeding herds, those that initiate the infection chain. A later section displays the performance of the proposed grid graph-partitioning scheme and compares it to the previous approach.

6.1. Experimental Setup

The benchmarks and evaluations carried out in this study were performed on a cluster of 30 HP Z420 servers (16-core Xeon E5-2560V2 @ 2.60 GHz, 32 GB RAM, 1 TB disk). Distributed computations were executed on Apache Spark GraphX version 2.0 with OpenJDK JVM, version 1.8.0_131 and Scala version 2.10.4. Each host was configured with Fedora 25 (64-bit Linux kernel 4.5.7). The epidemiological test dataset from Colorado, USA, was used, and it was distributed across the Hadoop cluster (version 2.7.3), totaling 3.2 million scenario iterations aggregating 6.26 TB data.

6.2. PageRank vs. Super-spreading Analysis

This part of the experiment depicts the analysis of the inclusion of super-spreaders in the composition of highly influential herds identified using PageRank algorithm. Following the work presented in Shah et al. [15], 3747 herds were identified as probable super-spreaders from the subject dataset. Afterwards, cardinality of the intersection set between top n PageRank valued herds and the 3747 super-spreaders was calculated for $n \in \{50, 100, 200, \ldots, 18800\}$. The ROC curve for this experiment was plotted, shown in Figure 6.1. According to the curve, the experiment resulted in high accuracy, meaning super-spreaders account for a considerably large portion of the overall set of herds with high PageRank values, which are highly influential. The reason behind

this result is that both groups infect a higher number of herds on average; as mentioned in [15]. The degree of local infection contributes most when classifying a herd as a super-spreader, and herds with high PageRank values tend to infect a higher number of herds overall, as mentioned in Chapter 4. Moreover, it can be observed that the likelihood ratio is decreasing while moving along the horizontal axis. The part of curve with a high likelihood ratio refers to herds with high influence values, whereas the other part of the curve refers to its counterpart.

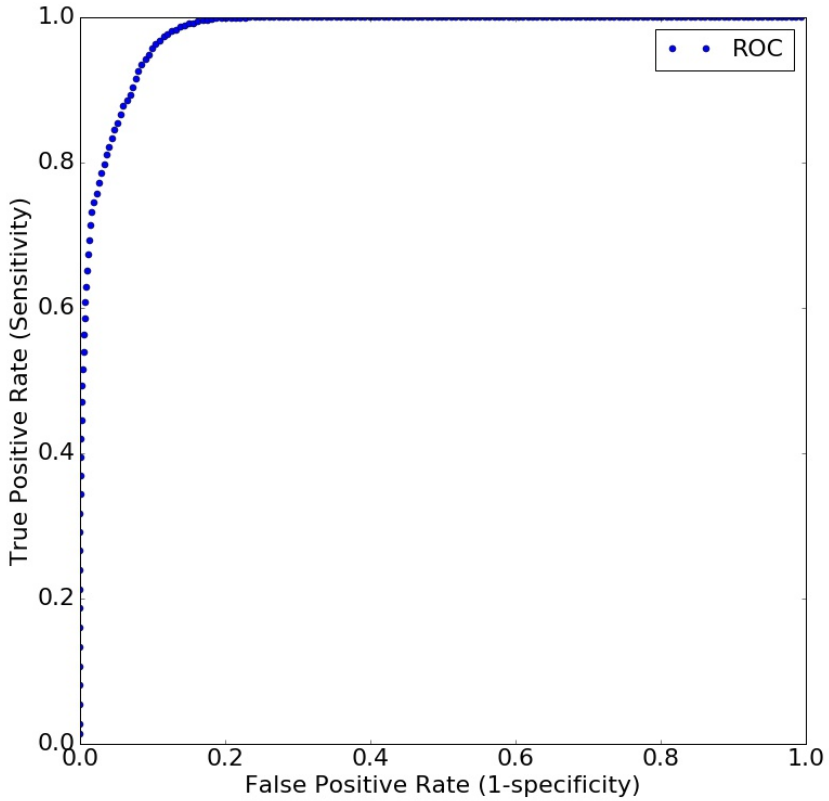


Figure 6.1. High PageRank Valued Herds vs. Super-spreadering Events

A two-sample t-test was performed to determine whether the tendency to include super-spreaders in high- and low-PageRank herds was statistically significant. In this evaluation, the top 20% of PageRanked herds (likely super-spreaders) were assessed with the next 20%. To conduct the t-test, 40 data points were generated by randomly selecting 1000 herds from each set and

the count of super-spreaders was recorded. This experiment revealed a significant difference between herds with high PageRank values ($\bar{x_1} = 839.93$, $\bar{s_1} = 11.26$) and herds with lower PageRank values ($\bar{x_2} = 192.5$, $\bar{s_2} = 9.9$); $t(76.72) = 1.84$, $p = 0.03452$ for $\mu_0 = 643$. These results suggest that the mean number of super-spreaders found in both groups is notably different.

6.3. PageRank vs. Seeding Analysis

This experiment analyzed the involvement of seeder herds, the ones that are infected by the set of initially infected herds, in the evolution of super-spreaders. As described in Chapter 4, initially infected herds have been removed from the infection propagation pairs, and the rest of the data has been collected for analysis. Over the 3.2 million iterations, 6504 distinct seeders were found. The same experimental procedure as in the previous study was performed in order to visualize the involvement of seeder herds in the composition of highly influential herds. Therefore, the number of herds having the top n PageRank values, who were among the seeders, was calculated for n *in* {50, 100, 200, ... , 18800} and plotted on the curve, which is shown in Figure 6.2. Initially, the curve shows a small peak, which is later followed by a monotonically increasing curve. The area under the curve is much smaller compared to the curve generated in the previous experiment performed on super-spreaders. This result suggests that seeders do not contribute to the composition of highly influential herds as much as the super-spreaders. There are likely two reasons for this: First, among the 6504 seeder herds, most are classified as seeders a limited number of times in the overall dataset of 3.2 million simulated outbreaks, resulting in a lower number of overall infections. Second, seeders often infect herds with a low PageRank value, resulting in little contribution towards their own influence.

To study the involvement of super-spreaders and seeders combined as a single group, the union of two sets is computed to compare with highly influential herds derived from PageRank values. Figure 6.3 shows the size of each of these sets based on the top n PageRank values. This demonstrates that about 3000 of the top herds are either super-spreaders or seeders (with the

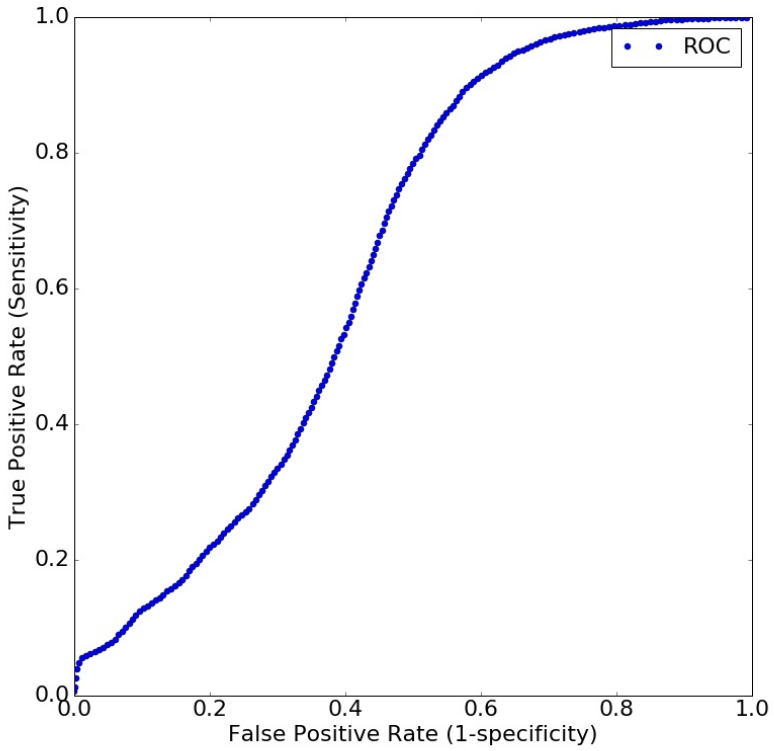


Figure 6.2. High PageRank Valued Herds vs. Seeders

majority being super- spreaders), as the initial portion of the curve overlaps with the identity line. After all the super-spreaders are considered ($n = 7100$), the union set follows the shape of the seeder plot. This demonstrates that herds with the highest influence are largely super-spreaders.

The true Positive Rates (TPR) and False Positive Rates (FPR) used to create the ROC curves in the previous experiments were calculated using the following formula:

$$TPR_n = \frac{NI_n}{T_p}; FPR_n = \frac{n - NI_n}{T_n} \tag{3}$$

where

NI_n = Intersection of super-spreaders or seeders with the top n high PageRank valued herds,

T_p = Total number of super-spreaders or seeders, and

T_n = Total number of non-super-spreaders or non-seeders.

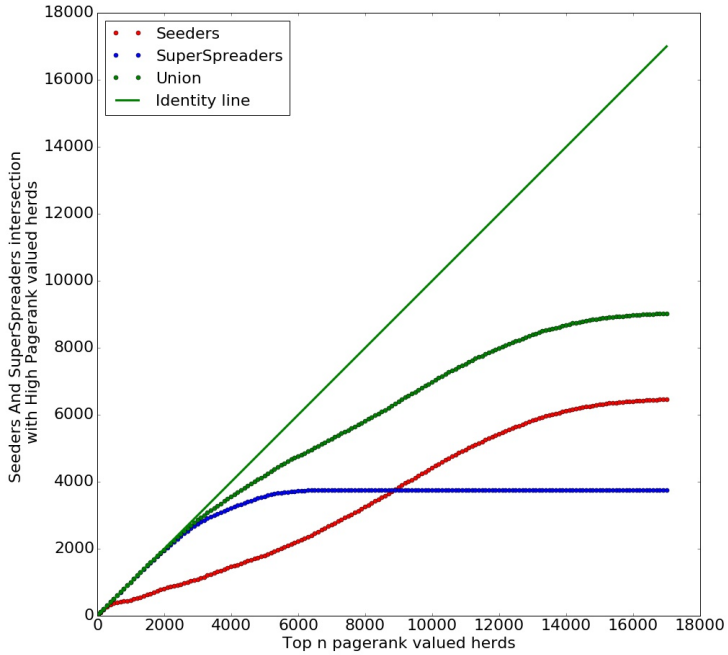


Figure 6.3. Seeders, Super-spreaders, and their Union Based on the Top n PageRanked Herds

6.4. Scalability Evaluation

In this experiment, the time taken by the Apache Spark GraphX framework to compute PageRank values of herds in the disease transmission network for various combinations of data and cluster sizes was measured. From the 100,000 simulation outputs from our Colorado dataset, disease transmission information in the form of infection propagation pairs was extracted, and the PageRank implementation was executed for 25 iterations. During this evaluation, cluster sizes with a varying number of machines were considered, each of which was accountable for four Spark workers. Figure 6.4 demonstrates the results of this benchmark; the vertical axis contains the time taken to perform the computation, and dataset sizes are presented on the horizontal axis. Clusters of 10 and 20 machines exhibited similar execution times due to resource constraints that increased synchronization delays between stages, but the cluster of 30 machines improved computation times by about 25% for the full-sized dataset, which is shown in Figure 6.4.

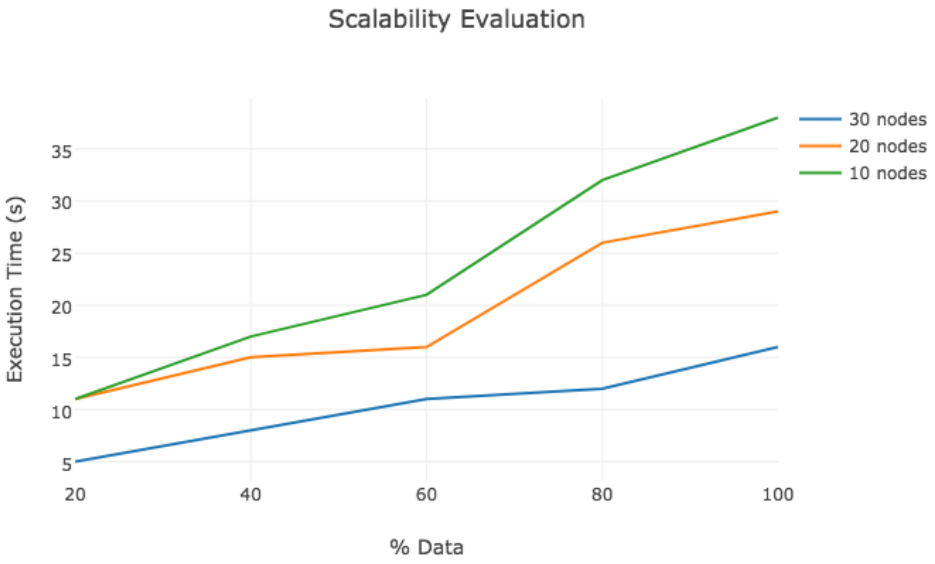
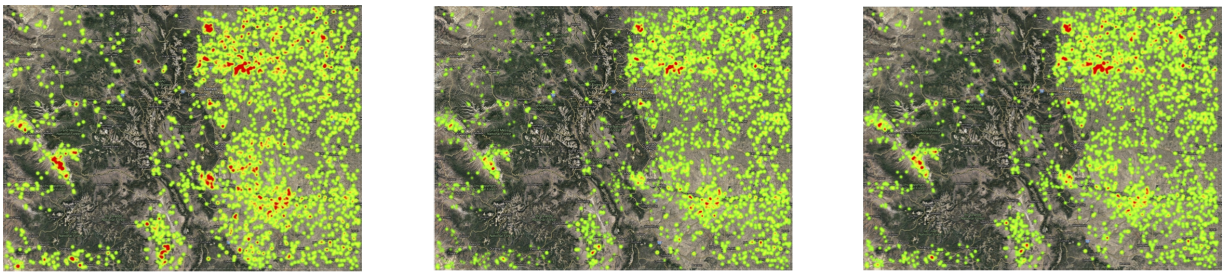


Figure 6.4. Scalability Evaluation

6.5. GeoSpatial Analysis

Figure 6.5 depicts three heat maps, each of which was generated using a different analytics approach to identify highly influential herds along with their geospatial locations in Colorado. Herds with higher influence are highlighted by brighter shades of red, whereas less influential herds are drawn in progressively darker shades of green. Note that these visualizations are based on the top 20% of the herds in the dataset to increase the level of contrast between premises. Three notable clusters can be seen in each of the subfigures, one in the mid-left, and another two near the top and bottom-right. Figure 6.5a was generated using PageRank values of herds, and the premise contribution to the overall infection (contherdID) is shown in Figure 6.5b [15]. Note that both heat maps are similar, indicating that the super-spreaders detected by herd contributions are a subcategory of the influential premises found via PageRank. On the other hand, Figure 6.5c is formed using the Machine Learning model proposed by Shah et al. [15], in which the darker the area in the heat map, the greater the confidence of the corresponding herd's being classified as a super-spreader.



(a) Top 20% High PageRank Valued Premises

(b) Premises' Contribution to Overall Infection

(c) Classified Super-spreaders Using the SVM Model

Figure 6.5. GeoSpatial Information of Highly Influential Herds Based on Three Different Analytics approaches

The rest of this chapter evaluates the extension of the 2D graph-partitioning strategy provided by Apache Spark GraphX, which was described in the previous chapter and demonstrated in the comparison with the existing implementation. This section covers the comparison in terms of graph ingression time, vertex replication factor, execution time of the PageRank algorithm for a predefined number of iterations, and workload distribution. All of the above experiments were conducted on the inverted graph generated from the subject dataset, which is the DTN generated from NAADSM simulation outputs of Colorado dataset, and comprised of 18,890 vertices and 1,682,361 edges. All of the results were collected for the same set of varied numbers of partitions, that is {10, 25, 50, 75, 100, 125, 144, 200}. As this set contains perfect as well as non-perfect squared numbers, the evaluation embodies all aspects of the candidate algorithms.

6.6. Graph Ingression Time

As mentioned in Chapter 2, graph ingression time refers to the time it takes to partition and load the graph into memory. Conceptually, it's the time taken by the partitioning algorithm starting from dividing the graph and providing the meaning to each of the subgraphs up to loading each of those into memory for further computation. Clearly, a partitioning scheme may affect the ingression time adversely, so it is worthwhile to verify the actual ingression time and compare it to the previous implementation. Figure 6.6 shows the comparison of ingression time between the two partitioning

algorithms for varying of numbers of partitions. The enhanced algorithm delivered nearly the same results as the *EdgePartition2D* scheme. This observation is reasonable as both the algorithms are implemented using stream-based hashing techniques. They are capable of placing an edge or a vertex in an appropriate partition without having any prior knowledge of other placements.

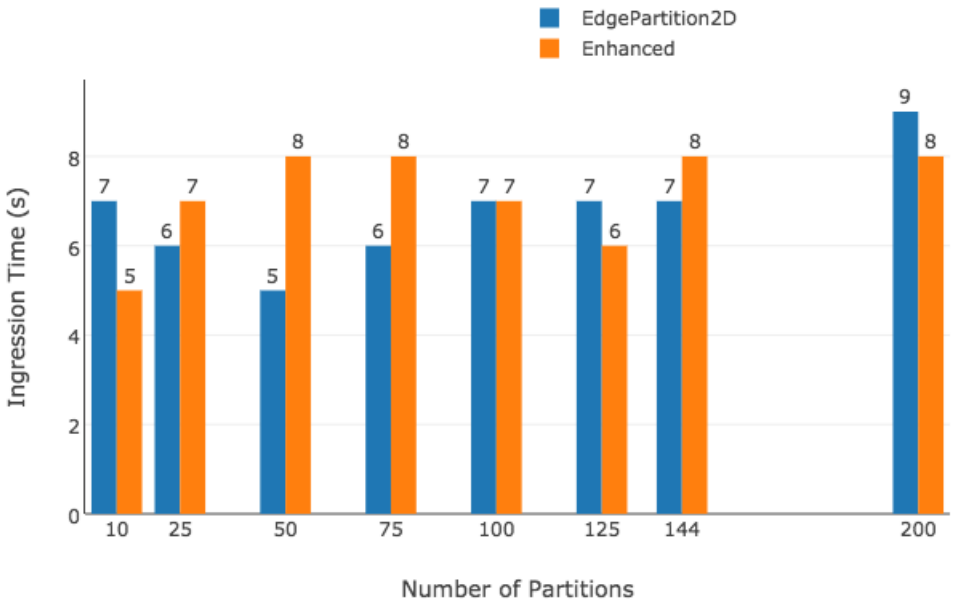


Figure 6.6. Graph Ingression Time

6.7. Replication Factor

The vertex replication factor is a single representative value for the graph that describes the amount of replication incurred by the partitioning algorithm. As it is directly associated with the amount of communication, it is the key metric that decides the performance of any partitioning algorithm. Obviously, a smaller replication factor implies less communication and better performance. Hence, the ideal partitioning algorithm tries to improve itself by finding the avenues that reduce the replication factor. Figure 6.7 shows a comparison between the original and the enhanced partitioning algorithms in terms of the resulting eventual mean vertex replication factor for a set of varying numbers of partitions. Undoubtedly, the enhanced version demonstrated better replication

factor results than the standard *EdgePartition2D* scheme. It is worth noting that the difference in the mean replication factor for any particular observation is more than one. The reason behind this behavior is that the enhanced algorithm brings down the upper bound on vertex replication from $2\sqrt{n}-1$ to $2\sqrt{n}-2$, n being the number of partitions, which is explained theoretically in Chapter 5. Subsequently, the mean replication factor for any observation is less than the corresponding upper bound. For example, for $n = 100$, the mean replication factor observed is 14.3807, which is less than $2\sqrt{100}-2 = 18$.

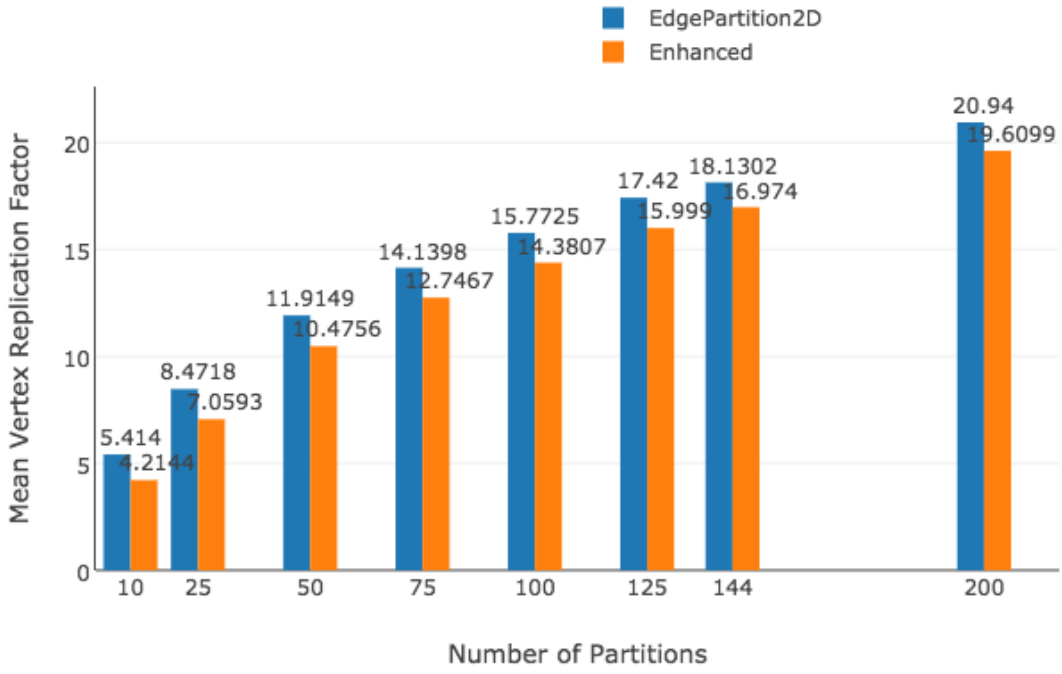


Figure 6.7. Vertex Replication Factor

In order to determine whether the differences between mean replication factor of the *EdgePartition2D* and that of the enhanced version are statistically significant, the Wilcoxon signed-rank test was performed as the distribution of the difference in mean replication factor was not normal (*p-value=0.03744* for Shapiro-Wilk Normality Test). The Wilcoxon signed-rank test demonstrated that the differences between mean replication factor of both the approaches are statistically significant *(Z=-2.5205, p=0.007812)*. Moreover, the 95% Bootstrap Studentized Confidence Interval is

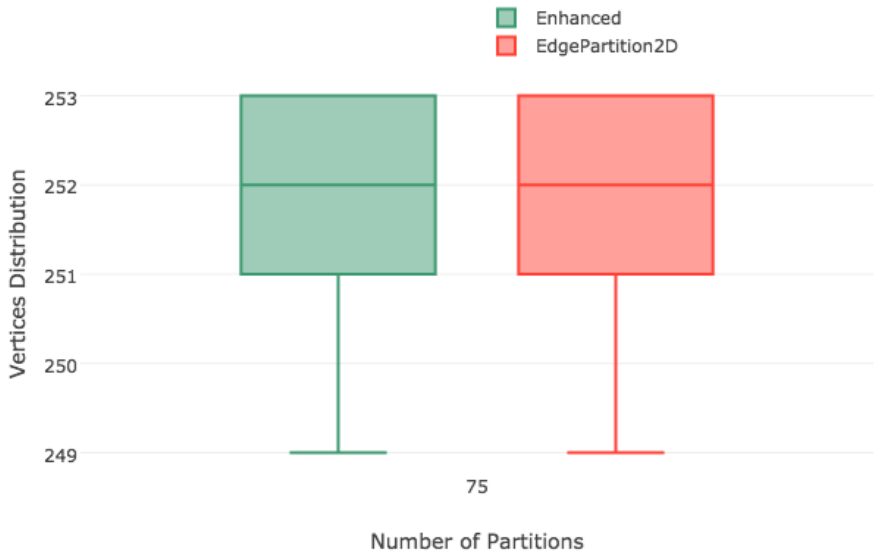
(-1.414, -1.141) which indicates that the mean replication factor in enhanced version is at least one less than that in *EdgePartition2D*.

6.8. Load Balancing

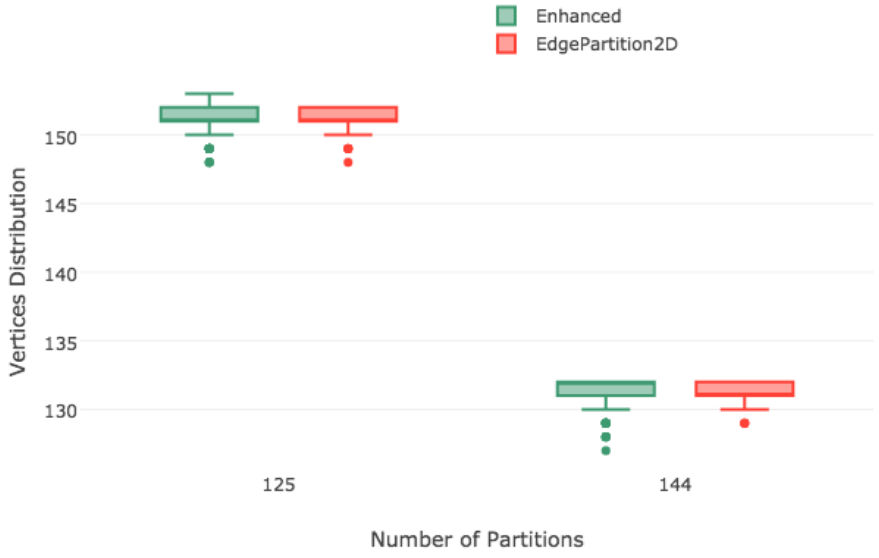
Load Balancing refers to the distribution of graph properties (vertices and edges) among partitions. A key issue that affects this metric is that the workload distribution and the replication factor behave in a completely opposite manner. Oftentimes, attempts to improve one result in ruining the other. Edge-cut partitioning is a classic example of this issue. In an attempt to improve the replication factor for high-degree vertices in Power-law graphs, this strategy delivers unbalanced partitions in terms of edge-distribution. In contrast, an ideal scheme should have control of both of these factors. Load balancing is measured either by the statistical properties of a distribution, which include the mean, median, and standard deviation, or using side-by-side boxplots of the distribution. Generating plots help readers visualize the distributions, which is essential to comparing multiple approaches.

Figure 6.8 displays the distribution of vertices for both the approaches. The observed similarity between them along with the reduction in upper bound of the vertex replication factor is the prime reason for improvement in the performance of the later approach compared to the previous. Figure 6.8a depicts the comparison in the vertices distribution for $n = 75$, and Figure 6.8b demonstrates the variation in distributions for $n = 125$ and $n = 144$. Evidently, the distribution of vertices remains nearly the same with varying numbers of partitions. It can be observed in figure 6.8b that, even for the perfect square number of partitions, vertices are distributed in a balanced fashion rather than placed only in the diagonal partitions of the grid. As mentioned in the previous chapter, this issue was handled by employing different hashing techniques for the source and the destination vertices.

Figures 6.9a and 6.9b display the edge distribution for varying numbers of partitions, including both the perfect and the non-perfect square. Apparently, edge distributions remain similar for



(a) $n = 75$

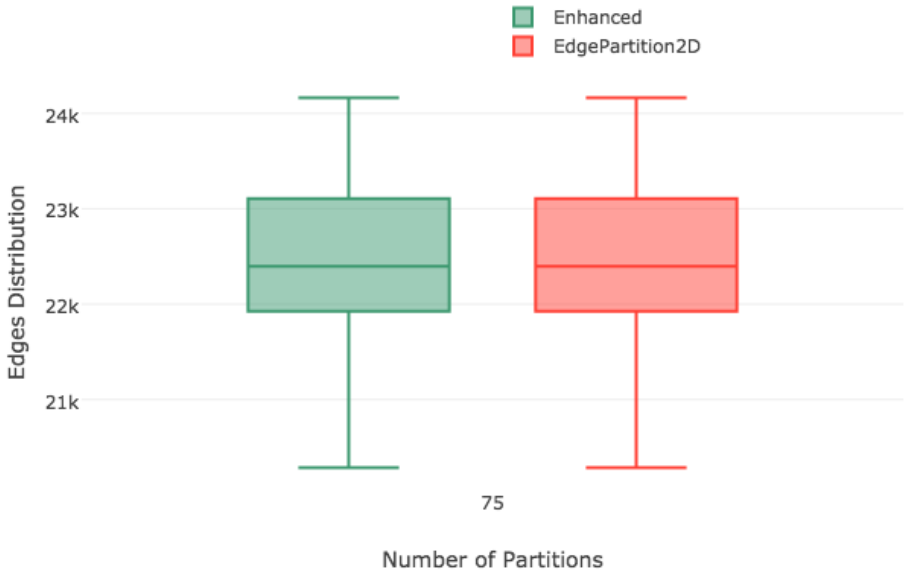


(b) $n = \{125, 144\}$

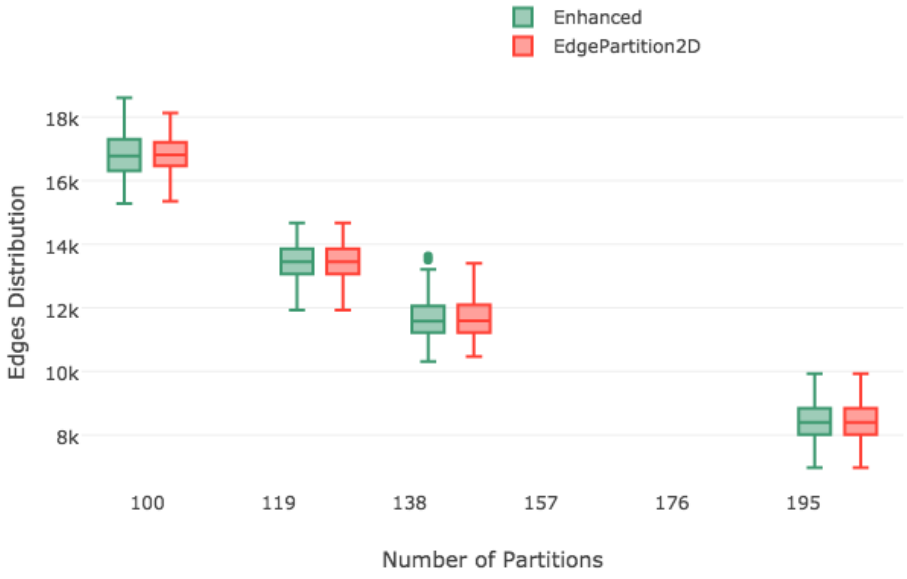
Figure 6.8. Vertices Distribution

same number of partitions. In fact, the edge partitioning logic for the non-perfect square number of partitions is untouched by the new algorithm. Consequently, the placement and subsequent distribution of edges also remains the same for that case. Moreover, for the other case, only the hashing of the source vertex was changed, not that of the destination vertex, which ensures that

the row of the partition remains the same. This balanced distribution of vertices and edges along with the reduced mean replication factor allows the new algorithm to perform better.



(a) $n = 75$



(b) $n = \{100, 125, 144, 200\}$

Figure 6.9. Edges Distribution

6.9. Execution Time

Execution time refers to the entire time starting from the creation of the graph from raw data representation to the time that it takes for the algorithm to finish. Clearly, this metric is affected by all three metrics mentioned above. Therefore, it is mandatory to verify the actual execution time required by both algorithms. Figure 6.10 shows the execution times required by each algorithm, and it can be observed that the new algorithm completed earlier or at least at the same time as the *EdgePartition2D* in all cases. The primary reason for this result is that the reduction in the mean replication factor of a vertex allows the algorithm to reduce communication and complete the execution faster.

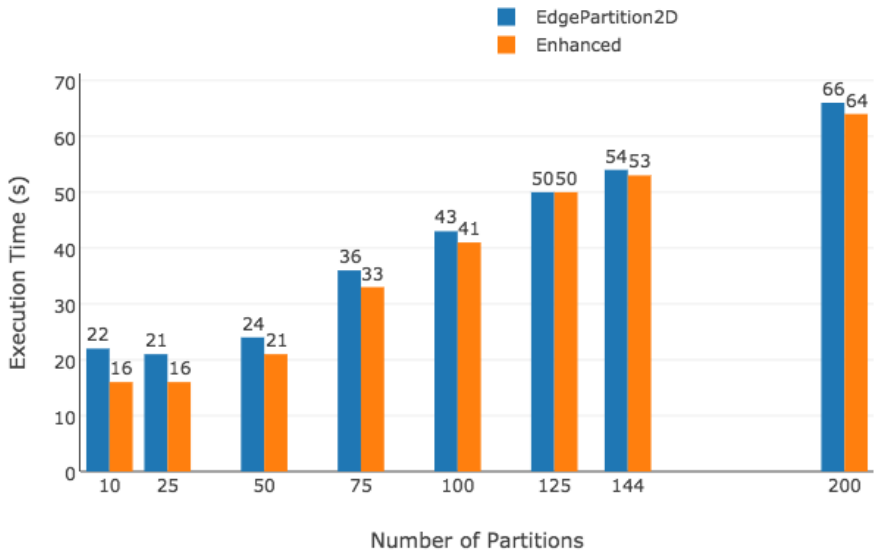


Figure 6.10. Execution Time

A Paired Samples t-test was performed to compare the *EdgePartition2D* and the enhanced version of the same in terms of the execution time. It represented significant difference in the execution time of the enhanced version *(M=36.75, SD=17.04)* and that of the *EdgePartition2D (M=39.5,SD=15.56)*; $t(7)=3.671$, $p < 0.005$. These results suggest that the execution time in the enhanced version has decreased compared to that in the *EdgePartition2D*.

CHAPTER 7

Conclusion and Future Research Directions

This thesis delivers an analytical approach to determining influence measures of premises from simulated voluminous epidemiology data that facilitate the effective use of limited resources during a disease outbreak. This work demonstrates the concise representation of the copious simulated data using the Disease Transmission Network (DTN), which is a weighted, directed graph that encapsulates infection interactions while retaining the dynamics of the disease outbreak. The influence of premises was computed by leveraging the PageRank algorithm, which gained popularity by sorting webpages based on their importance in the webgraph and later found application in a variety of domains. Moreover, the analysis was performed on the extended version of the distributed graph processing framework, Apache Spark GraphX, which improves performance by minimizing the communication metric. This study answers the research questions raised while designing the solution.

(1) **What data structure(s) allow the representation of disease spread interactions for analysis?** To achieve effective analysis with reasonable latency, entire chains of infections were extracted from the output dataset, and a graph-based data structure, the Disease Transmission Network (DTN), was constructed that represents a holistic view of disease transmissions by maintaining the probability of infections between each herd pair. The DTN is a compact data structure that is less than 0.002% of the original dataset size. Since infections between herds are observed over 3.2 million iteration outputs, maintaining this pairwise probability with the DTN reduces the number of I/O accesses (encompassing both disk and network I/O) to the dataset significantly.

(2) **How is the influence of each premise measured?** The PageRank algorithm was leveraged to estimate the influence of each herd in the DTN. The PageRank value associated with a premise represents the probability that it contributes to a random infection chain. The results of

this study indicate that the premises with higher influential value tend to have super-spreading characteristics.

(3) **How is the analysis achieved at scale in the least time?** The analysis was performed on an environment that consisted of the reliable and fault tolerant distributed file system, HDFS, and an extended version of a distributed in-memory graph-parallel system, Apache Spark GraphX. The performance of GraphX is enhanced by extending the *EdgePartition2D* graph-partitioning strategy such that the extension reduces the upper bound on mean vertex replication factor by one, hence minimizes the overall execution time.

As part of future work, the DTN can be extended to accommodate other features such as types of herds, time-related information, and verifying the quality of influence measures. Moreover, the graph-parallel system can be further extended to employ a graph-partitioning strategy specifically designed for the updated version of the DTN. In addition to that, the support for incremental PageRank algorithm can be added such that any impacts on the relative influence measure resulting from changes in the DTN can be accommodated in real-time.

References

[1] E. Brooks-Pollock, M. de Jong, M. J. Keeling, D. Klinkenberg, and J. L. Wood, "Eight challenges in modelling infectious livestock diseases," *Epidemics*, vol. 10, pp. 1–5, 2015.

[2] D. Thompson, P. Muriel, D. Russell, P. Osborne, A. Bromley, M. Rowland, S. Creigh-Tyte, C. Brown *et al.*, "Economic costs of the foot and mouth disease outbreak in the united kingdom in 2001," *Revue scientifique et technique-Office international des epizooties*, vol. 21, no. 3, pp. 675–685, 2002.

[3] T. Knight-Jones and J. Rushton, "The economic impacts of foot and mouth disease–what are they, how big are they and where do they occur?" *Preventive veterinary medicine*, vol. 112, no. 3, pp. 161–173, 2013.

[4] N. R. Council *et al.*, *Workforce needs in veterinary medicine*. National Academies Press, 2013.

[5] I. CFSPH, HSEMD, "Animal disease emergencies local response, preparedness and planning," 2008. [Online]. Available: http://www.cfsph.iastate.edu/Animal_Response/English/pdf/A4_SPN_BusinessOverview.pdf

[6] M. J. Grubman and B. Baxt, "Foot-and-mouth disease," *Clinical microbiology reviews*, vol. 17, no. 2, pp. 465–493, 2004.

[7] J. C. Miller and J. M. Hyman, "Effective vaccination strategies for realistic social networks," *Physica A: Statistical Mechanics and its Applications*, vol. 386, no. 2, pp. 780–785, 2007.

[8] M. J. Keeling and P. Rohani, *Modeling infectious diseases in humans and animals*. Princeton University Press, 2008.

[9] N. Harvey, A. Reeves, M. A. Schoenbaum, F. J. Zagmutt-Vergara, C. Dubé, A. E. Hill, B. A. Corso, W. B. McNab, C. I. Cartwright, and M. D. Salman, "The north american animal disease spread model: A simulation model to assist decision making in evaluating animal disease incursions," *Preventive veterinary medicine*, vol. 82, no. 3, pp. 176–197, 2007.

[10] D. L. Pendell, J. Leatherman, T. C. Schroeder, and G. S. Alward, "The economic impacts of a foot-and-mouth disease outbreak: a regional analysis," *Journal of agricultural and applied economics*, vol. 39, no. s1, pp. 19–33, 2007.

[11] C. Green, T. Whiting, G. Duizer, D. Douma, H. Kloeze, W. Lees, and A. Reeves, "Simulation modeling of alternative control strategies for an hpai outbreak using naadsm," in *Canadian Association of Veterinary Epidemiology Preventive Medicine (CAVEPM) Meeting*, 2010.

[12] K. Portacci, A. Reeves, B. Corso, and M. Salman, "Evaluation of vaccination strategies for an outbreak of pseudorabies virus in us commercial swine using the naadsm," *ISVEE*, vol. 12, p. 78, 2009.

[13] Z. Sui, N. Harvey, and S. Pallickara, "On the distributed orchestration of stochastic discrete event simulations," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 11, pp. 1889–1907, 2014.

[14] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[15] N. Shah, H. Shah, M. Malensek, S. L. Pallickara, and S. Pallickara, "Network analysis for identifying and characterizing disease outbreak influence from voluminous epidemiology data," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1222–1231.

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. IEEE, 2010, pp. 1–10.

[17] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013, p. 2.

[18] Z. Sui, M. Malensek, N. Harvey, and S. Pallickara, "Autonomous orchestration of distributed discrete event simulations in the presence of resource uncertainty," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 3, p. 18, 2015.

[19] W. Budgaga, M. Malensek, S. Pallickara, N. Harvey, F. J. Breidt, and S. Pallickara, "Predictive analytics using statistical, learning, and ensemble methods to support real-time exploration of discrete event simulations," *Future Generation Computer Systems*, vol. 56, pp. 360–374, 2016.

[20] M. Malensek, W. Budgaga, S. Pallickara, N. Harvey, F. J. Breidt, and S. Pallickara, "Using distributed analytics to enable real-time exploration of discrete event simulations," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2014, pp. 49–58.

[21] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges: Graph processing at facebook-scale," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1804–1815, 2015.

[22] 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.

[23] A. Giraph, "Apache giraph," 2011. [Online]. Available: http://giraph.apache.org/

[24] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.

[25] R. R. McCune, T. Weninger, and G. Madey, "Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, p. 25, 2015.

[26] K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2004, pp. 120–124.

[27] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs." in *OSDI*, vol. 12, no. 1, 2012, p. 2.

[28] S. Funk, M. Salathé, and V. A. Jansen, "Modelling the influence of human behaviour on the spread of infectious diseases: a review," *Journal of the Royal Society Interface*, p. rsif20100142, 2010.

[29] S.-J. Paine, P. H. Gander, and N. Travier, "The epidemiology of morningness/eveningness: influence of age, gender, ethnicity, and socioeconomic factors in adults (30-49 years)," *Journal of biological rhythms*, vol. 21, no. 1, pp. 68–76, 2006.

[30] B. Xiang, Q. Liu, E. Chen, H. Xiong, Y. Zheng, and Y. Yang, "Pagerank with priors: An influence propagation perspective," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[31] C. C. Aggarwal, A. Khan, and X. Yan, "On flow authority discovery in social networks," in *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 2011, pp. 522–533.

[32] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 137–146.

[33] B. Hajian and T. White, "Modelling influence in a social network: Metrics and evaluation," in *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 497–500.

[34] M. Cha, H. Haddadi, F. Benevenuto, and P. K. Gummadi, "Measuring user influence in twitter: The million follower fallacy." *Icwsm*, vol. 10, no. 10-17, p. 30, 2010.

[35] J. Goldenberg, B. Libai, and E. Muller, "Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata," *Academy of Marketing Science Review*, vol. 2001, p. 1, 2001.

[36] ——, "Talk of the network: A complex systems look at the underlying process of word-of-mouth," *Marketing letters*, vol. 12, no. 3, pp. 211–223, 2001.

[37] A. Khrabrov and G. Cybenko, "Discovering influence in communication networks using dynamic graph analysis," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 288–294.

[38] M. Malensek, S. Pallickara, and S. Pallickara, "Fast, ad hoc query evaluations over multidimensional geospatial datasets," *IEEE Transactions on Cloud Computing*, 2015.

[39] ——, "Analytic queries over geospatial time-series data using distributed hash tables," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1408–1422, 2016.

[40] ——, "Evaluating geospatial geometry and proximity queries using distributed hash tables," *Computing in Science & Engineering*, vol. 16, no. 4, pp. 53–61, 2014.

[41] W. Budgaga, M. Malensek, S. Lee Pallickara, and S. Pallickara, "A framework for scalable real-time anomaly detection over voluminous, geospatial data streams," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, 2017.

[42] J. C. Kachikaran Arulswamy and S. L. Pallickara, "Columbus: Enabling scalable scientific workflows for fast evolving spatio-temporal sensor data." in *Proceedings of the 14th IEEE International Conference of Service Computing (IEEE SCC)*, 2017.

[43] S. L. Pallickara and M. Pierce, "Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 285–292.

[44] S. L. Pallickara, M. Pierce, Q. Dong, and C. Kong, "Enabling large scale scientific computations for expressed sequence tag sequencing over grid and cloud computing clusters," in *PPAM 2009 EIGHTH INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING AND APPLIED MATHEMATICS Wroclaw, Poland*, 2009.

[45] H. P. Sajjad, A. H. Payberah, F. Rahimian, V. Vlassov, and S. Haridi, "Boosting vertex-cut partitioning for streaming graphs," in *Big Data (BigData Congress), 2016 IEEE International Congress on*. IEEE, 2016, pp. 1–8.

[46] H. Halberstam and R. Laxton, "Perfect difference sets," in *Proceedings of the Glasgow Mathematical Association*, vol. 6, no. 04. Cambridge Univ Press, 1964, pp. 177–184.

[47] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni, "Hdrf: Stream-based partitioning for power-law graphs," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 243–252.

[48] C. Xie, L. Yan, W.-J. Li, and Z. Zhang, "Distributed power-law graph computing: Theoretical and empirical analysis," in *Advances in Neural Information Processing Systems*, 2014, pp. 1673–1681.

[49] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 1.

[50] S. Pallickara, J. Ekanayake, and G. Fox, "Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[51] T. Buddhika and S. Pallickara, "Neptune: Real time stream processing for internet of things and sensing environments," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 1143–1152.

[52] S. Pallickara and G. Fox, "On the matching of events in distributed brokering systems." in *ITCC (2)*, 2004, pp. 68–76.

[53] G. Fox, S. Pallickara, and X. Rao, "Towards enabling peer-to-peer grids," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 7-8, pp. 1109–1131, 2005.

[54] N. Jain, G. Liao, and T. L. Willke, "Graphbuilder: scalable graph etl framework," in *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013, p. 4.

[55] GitHub, "Apache spark graphx source code," 2015. [Online]. Available: https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/PartitionStrategy.scala

[56] Attribution for Documents Icon[1] and Magnifier Icon[2] in figure 4.1

[57] Attribution for NAADSM logo[3], HDFS logo[4], and GraphX logo[5] in figure 4.1

[1]Made by Freepik (http://www.freepik.com) from Flaticon (http://www.flaticon.com) is licensed by Flaticon Basic License (http://file000.flaticon.com/downloads/license/license.pdf)

[2]Made by Iconfinder (https://www.iconfinder.com/icons) is licensed by Basic license (https://www.iconfinder.com/licenses/basic)

[3]Made by NAADSM Development Team (http://www.naadsm.org/) is licensed by Creative Commons License 3.0(https://creativecommons.org/licenses/by-nc-sa/3.0/)

[4]Made by Hadoop 2.7.3 Development Team (https://hadoop.apache.org/docs/r2.7.3/) from The Apache Software Foundation (https://www.apache.org/) is licensed by Apache License 2.0(https://www.apache.org/licenses/LICENSE-2.0)

[5]Made by Spark GraphX Development Team (https://spark.apache.org/graphx/) from The Apache Software Foundation (https://www.apache.org/) is licensed by Apache License 2.0(https://www.apache.org/licenses/LICENSE-2.0)