Received: 00 Month 0000 | Revised: 00 Month 0000 | Accepted: 00 Month 0000

DOI: xxx/xxxx

RESEARCH ARTICLE

Scalable Network Analytics for Characterization of Outbreak Influence in Voluminous Epidemiology Datasets

Naman Shah[1] | Matthew Malensek[1,2] | Harshil Shah[1] | Shrideep Pallickara[1] | Sangmi Lee Pallickara[1]

[1]Department of Computer Science, Colorado State University, Fort Collins, Colorado, USA
[2]Department of Computer Science, University of San Francisco, California, USA

Correspondence
Matthew Malensek, Email: mmalensek@usfca.edu

Summary

Planning for large-scale epidemiological outbreaks in livestock populations 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.

In this study, we set out to achieve two goals: (1) providing a distributed framework for modeling disease transmission at scale using Spark, including improvements to the default GraphX partitioning strategy, and (2) giving planners and epidemiologists a means to analyze interactions between entities (herds) during simulated disease outbreaks. Using our *disease transmission network* (DTN), planners or analysts can isolate herds that have a disproportionate effect on epidemiological outcomes, enabling effective allocation of limited resources such as vaccinations and field personnel. We use a representative dataset to verify our approach, and optimized the underlying graph partitioning algorithm to ensure the system will scale with increases in the dataset size or number of participating machines. Our analysis includes identification of influential herds as well as the creation of machine learning models for accurate classifications that generalize to other datasets.

KEYWORDS:
Epidemiological network analysis, distributed analytics, disease spread classification, distributed graph partitioning, super-spreading events

1 | INTRODUCTION

According to the Food and Agricultural 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). Effective planning and response to infectious threats in livestock are critical for the ecological system, the global economy, and human health in the case of zoonotic diseases (such as swine flu) that exhibit cross-species transmission. There have been significant efforts in the epidemiological modeling community to understand and predict the distribution of disease within a herd as well as transmission between herds (2). Epidemiological models, often expressed as stochastic discrete event simulations, involve hundreds to thousands of input parameters and tend to be compute-intensive.

In this study, we consider the North American Animal Disease Spread Model (NAADSM), which has been vetted by over 300 epidemiologists and veterinarians and is one of the key tools used by the US Department of Agriculture to plan for disease incursions (3). NAADSM can be used to model foot-and-mouth disease (FMD), highly pathogenic avian influenza, swine flu, pseudorabies, and more (4, 5, 6). The holistic, system-based simulation approach in NAADSM considers disease biology parameters including transmission via airborne or direct contact, control measures (such as vaccinations), effectiveness of vaccines,

quarantines, shipments, and veterinarian visits. Simulated datasets are an easily-accessible stand-in for real-world data in studies that involve disease spread and outbreak prevention. However, it is worth noting that drawing direct conclusions from such datasets is often difficult and must be applied alongside a series of other planning, prevention, and response protocols that strike a balance between data-based and theory-based approaches (7, 8). These outbreaks tend to be devastating to the livestock populations and local economies when they do happen, motivating continued study across disciplines.

In stochastic simulations such as NAADSM, 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. Additionally, these studies often include "what-if" analysis where several parameters are adjusted to determine their impact on overall disease spread. Each adjustment to these parameters results in a new *variant* of an outbreak scenario. Several cycles of what-if analysis, combined with the number of possible input parameters and the need for multiple simulation iterations produces voluminous output datasets. While efforts have been made to reduce the size of these datasets through prediction, their computational and storage demands remain high (9). Key outputs used during planning include the disease duration, number of infections, and depletion of vaccine stockpiles. While this study targets livestock disease outbreaks, the methodology that we describe is broadly applicable to systems where entities are organized into large networks and the spread of information — be it pathogens, ideas, or traffic movements — is based on relationships between entities.

One of the primary concerns during disease outbreak planning and prevention is allocating limited resources. Our goal in this effort is to identify entities (herds) that could contribute disproportionately to disease spread; i.e., once a particular herd is infected, the overall disease duration, total number of infections, and the probability of the disease becoming endemic are all high. Identifying such herds allows limited resources such as vaccines, field personnel, and biosurveillance to be allocated more effectively and in a targeted fashion. This involves analyzing voluminous data from simulation runs and tracking disease evolution over time. Pinpointing *highly influential herds* that contribute disproportionately to outbreaks is key when developing an effective response plan.

1.1 | Scientific Challenges

Timely identification and characterization of influential herds introduces a set of unique challenges:

1. ***Dataset Size***: Epidemiological state is dispersed over a large number of files (3.2 million in our primary dataset). Each simulated time step produces an output file containing a variety of simulation data that must be processed to capture disease spread over time (6.26 TB of files in our primary dataset, with an additional 8 TB from secondary datasets for a total of 14.26 TB).
2. ***Timeliness***: Our algorithms and analysis workflows must execute in parallel across a cluster of computing resources to ensure timely results. Given the data volumes and disk I/O costs involved, repeated sweeps over the dataset would introduce significant delays in analysis.
3. ***Scalability***: The proposed approach must scale with increases in the number of herds and interconnectivity between them. This ensures that the methodology is applicable in other scenarios.
4. ***Accuracy and Interpretability***: Our analysis must be reasonably accurate, and support interpretability by explaining why a herd is considered highly influential. This is critical for fine-tuning outbreak responses.

1.2 | Research Questions

Research questions that we explore in this study include the following:

RQ1 *What data structure(s) allow us to represent disease spread interactions for analysis?* Specifically, we must capture infection information from the simulation output and preserve the cumulative dynamics of disease spread. (§4.2)

RQ2 *How can we measure the influence of each herd?* This involves discovering the epidemic characteristics of influential herds as well as the features that comprise these characteristics, which enables interpretability and herd classification. (§4.3)

RQ3 *How can we enable the analysis at scale?* Given the data volumes at hand, we must avoid repeated sweeps over on-disk data and execute analysis concurrently on multiple machines. Specifically, our methodology must scale with increases in the number of herds, contacts, and machines available for analysis. (§4.4.5, §5)

RQ4 *Given a data model for disease transmission, how can we improve performance in a distributed setting?* This involves reducing the amount of time required to construct the data representation, avoiding communication latencies between participating hardware, and ensuring effective load balancing (§5).

1.3 | Overview of Approach

Our methodology for identification of influential herds in voluminous epidemiology data involves: (1) extracting relevant information needed for analysis from the dataset, (2) constructing a graph-based data structure, called the *disease transmission network* (DTN) to encode this information, (3) using the DTN for network analysis via the PageRank algorithm, and (4) identification and characterization of *super-spreaders* and *seeders*. Preprocessing and analysis tasks are expressed as distributed computations implemented using Apache Spark (10), with the dataset stored in HDFS (11). These tasks execute concurrently on multiple machines with data locality, and avoid making repeated disk accesses by performing analysis in main memory.

To further improve the performance of our approach, we adapted our algorithms to run within the Spark GraphX framework (12). We also provide extensions to the default GraphX partitioning algorithm, EdgePartition2D, to reduce the amount of network communication required during partitioning. This leads to better overall graph ingestion performance, which is vital in network-constrained environments or situations where the graph is frequently changed as more data becomes available.

Our epidemiology dataset encompasses multiple representative scenarios and iterations, which we process to extract and record millions of infection incidents. This includes tracking the number, source, destination, and duration of infections. This information is encoded in the disease transmission network. The DTN is a weighted, directed graph that summarizes the number of infections between herds; nodes within the DTN are herds and edges represent infection transmissions. The direction of traversals within the DTN varies depending on the algorithm underpinning the analysis.

Once generated, we analyze the DTN in multiple steps to identify and characterize highly influential herds. One avenue we leverage for analysis is the PageRank algorithm, which was originally used in the Google search engine to estimate the importance of web pages (13). In our study, we use PageRank to estimate the probability that a herd contributes to a random infection chain. We calculate PageRank values for each herd in the DTN; if a herd has a higher PageRank value, we consider the herd to be more influential in the disease outbreak.

Once we identify influential herds based on PageRank values, we perform further analysis to understand other epidemic characteristics such as classifying *super-spreaders* and *seeders*. In epidemiology, a super-spreader is a host that infects disproportionally more secondary contacts than other hosts. We use the Pareto Principle (14) to determine super-spreaders, and model the relationship between features extracted from the output dataset to classify the super-spreaders using support vector machines. On the other hand, seeders are hosts that are among the first to be infected. Besides global analysis using the DTN, we also allow identification of the most influential herd(s) on a local scale based on cross-herd reachability.

1.4 | Paper Contributions

This paper presents our approach for identifying and characterizing highly influential herds by analyzing voluminous epidemiology data. Our specific contributions include:

1. We have designed a graph-based data structure, the *disease transmission network*, that preserves cumulative dynamics of disease spread across space and time. The data structure supports traversals that are needed for analysis and characterization. This network, along with subsequent analysis, helps inform planning and response decisions in the case of infectious threats.

2. Novel identification of influential herds by harnessing and adapting the PageRank algorithm in the context of epidemiology, with support for interpretability of the analysis by identifying key features that characterize influential herds.

3. Classification of super-spreaders using machine learning models. The resulting models can be used to inform why a particular herd should be given priority during outbreak responses.

4. Our approach avoids repeated I/O passes over the datasets and compactly encodes results in the memory-resident disease transmission network. We also provide enhancements to the default graph partitioning algorithm in Spark GraphX to reduce network communication across partitions.

1.5 | Paper Organization

The rest of the paper is organized as follows. Section 2 outlines the simulation and dataset used in this study, followed by related work in Section 3. The first portion of our methodology in Section 4 describes our disease transmission network (DTN), our approach for identifying/classifying influential herds. Section 5 extends our methodology to improve both DTN construction performance and analysis latencies. Both methodology sections are concluded with detailed analysis and performance benchmarks. Finally, our conclusions and future research directions are described in Section 6.

1.6 | Paper Extensions

Since the publication of our previous work, *Network Analysis for Identifying and Characterizing Disease Outbreak Influence from Voluminous Epidemiology Data* (15), we have extended our system to incorporate graph partitioning optimizations that greatly speed up the analysis process. Our partitioning algorithm builds on Spark GraphX (12) and is broadly applicable to other types of disease spread modeling and graph-based representations. This extension includes a detailed discussion on the implementation of our graph partitioning strategy (Section 5), a thorough benchmark suite (Section 5.4), additional background and introductory material, and a survey of related graph partitioning techniques from the literature as well additional coverage of big data approaches (Sections 3.3 and 3.2). In total, this extension represents 50% new material.

2 | BACKGROUND

Our framework analyzes epidemiological data produced via simulated disease spread. We also leverage state-of-the-art technologies in distributed computation to perform our analysis. This section describes these components in detail and also provides information on the experimental setup used for the benchmarks in Sections 4.4 and 5.4.

2.1 | NAADSM

The North American Animal Disease Spread Model (NAADSM) is a stochastic simulation of disease outbreaks to aid in strategy development and decision making (3). In this model, groups of livestock, called *herds*, are the basis of the simulation. Disease spread between herds is influenced by production types, inter-group similarities (shipment rates, infection rates, etc.), relative locations, and distances between herds. When a herd is infected, it follows a natural cycle of disease states consisting of: susceptible, latent, sub-clinically infectious, clinically infectious, naturally immune, vaccine immune, and destroyed. This cycle can be interrupted by disease control strategies including quarantine, destruction and vaccination. Disease spread among herds can happen in any of three methods: direct contact, indirect contact, and airborne spread. Stochastic processes drive all operations in the model and 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*. Because the simulation is stochastic, it is generally run for several *iterations* (32 per scenario, in this study) to gain confidence in the output distributions. To reduce the overall execution time of the simulation, NAADSM can be parallelized over a cluster of computing resources in a fault-tolerant fashion (16).

2.2 | Dataset

Our 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 (9, 17). This process generated 100,000 *scenario variants* that were executed 32 times for a total of 3.2 million outputs (6.26 TB). In the interest of determining whether our models could generalize, we also used a similar dataset set in Iowa, USA, to bring the total dataset size up to 14.26 TB. In this particular scenario, a single initial herd is infected, with disease spread eventually encompassing tens of thousands of herds. The output of the simulation contains attributes representing the disease status of individual herds and how the infection spreads across herds within the network. These outputs also account for topological characteristics such as connectivity between the herds, proximity, and contact due to movements.

2.3 | System Components

We leverage the Spark framework (10) to provide scalable and fault-tolerant computing capabilities over a cluster of machines. Spark is used for writing applications to process large amounts of data which can be stored in distributed file systems such as HDFS, local file systems, or data streams, and includes functionality such as map, reduce, filter, and join. Compared to traditional MapReduce implementations, Spark allows in-memory, iterative computations. This is particularly beneficial for algorithms such as PageRank, and allows our analysis operations to avoid disk I/O unless absolutely necessary. We use Spark to generate disease transmission networks (DTNs) from our epidemiological simulation output dataset, as well as performing analysis of highly-influential herds based on the DTN. We also implemented the DTN under the Spark GraphX framework (12), with a performance comparison provided in Section 5.4.1. To facilitate distribution of files across the cluster and ensure data locality during computations, we used the Hadoop Distributed File System (HDFS) (11) to store our dataset and output files.

2.4 | Experimental Setup

The benchmarks and evaluations carried out in this study were performed on a cluster of 30 HP Z420 servers (8-core Xeon E5-2560V2 @ 2.60 GHz, 32 GB RAM, 1 TB disk). Distributed computations were executed on Spark 2.0 and Scala 2.11, with the OpenJDK JVM version 1.8.0_141. Each host was configured with Fedora 25 (Linux kernel 4.11.12). We used our epidemiological test dataset from Colorado, USA, which was distributed across the HDFS cluster (version 2.7.3), totaling 3.2 million scenario iterations and 6.26 TB of data. Additional scenarios set in Iowa, USA, were used to verify the performance of our classifications, which consumed another 8.0 TB of disk space for a total dataset size of 14.26 TB.

3 | RELATED WORK

Our solution for characterizing influential herds in disease outbreaks cross-cuts three areas of study: network analysis, big data, and graph partitioning. Herein we review related approaches.

3.1 | Network Analysis

Influential herds transmit 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. Analysis of influence in epidemiology has seen considerable study, with much of the work revolving around the various characteristics of infected entities and their impact on disease transmission (18, 19). However, these approaches generally examine standalone characteristics and not the underlying network or relationships that result from disease spread.

Social Network Analysis (SNA) focuses on human interactions in social networks, but can be applied to analyze animal epidemics as well. Considerable research has been conducted on influence in social networks (20, 21, 22, 23, 24, 25, 26). 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. 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. studies the influence of users in Twitter based on three metrics: *in-degree*, *retweets* and *mentions*. This approach uses Spearman's rank correlation coefficient to compare user influence, and evaluates the behavior of the three metrics for highly influential users (24). An approach outlined by Khrabrov and Cybenko (27) uses daily mentions of users on Twitter as a basis for calculating different rank metrics such as PageRank, DRank, and StarRank to determine influence.

Aggarwal et al. (21) proposes 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. 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 (21).

Substantial effort has been devoted to identifying hotspots that result in super-spreading events (SSEs). Lloyd-Smith et al. defines a protocol to identify super-spreaders, which is applicable in understanding SARS outbreaks (28). The protocol suggests that the mean number of secondary infections from a particular host follows a Poisson distribution and outliers are often accountable for super-spreading events. However, underestimation of the epidemic potential can occur when field observations of mean secondary infections are low (29). Fujie-Odagaki et al. focuses on intrinsically strong herd infectiousness and social connections (30). Our particular dataset, however, does not provide such information.

3.2 | Big Data Approaches

Distributed storage systems such as HDFS (11), Cassandra (31), HBase (32), and MongoDB (33) all provide the necessary interfaces to manage epidemiology data at scale. Combined with a computation engine such as Hadoop (34, 35, 36), these platforms are highly flexible, but do not provide network-specific analysis constructs. Additionally, the MapReduce paradigm does not readily support iterative (or loop-based) computations over datasets without extensions such as HaLoop (37). By leveraging the Spark framework, we can support iterative computation as well as in-memory working sets backed by resilient distributed datasets (RDDs) (38). This allows OLAP (online analytical processing) operations, contrasting with constrained-scale OLTP (online transaction processing) functionality found in traditional graph databases (39, 40).

Epidemiological big data analysis systems include Google Flu Trends (41), which uses web search data to model flu-like symptoms in user queries and leverages the correlation between medical searches and physician visits to estimate influenza activity across the United States. The system demonstrated faster results compared to traditional disease surveillance methods, but also suffered high-profile failures due to potential over-fitting, changes in the underlying Google search algorithm, and the opaque nature of the system (42). Scenarios such as these encouraged our use of several different metrics to strengthen the veracity of our findings. Galileo (43, 44, 45, 46) uses a graph-based indexing scheme to enable analysis between entities in multidimensional data, with support for spatial queries based on proximity, polygons, or administrative boundaries (47). SWAN (48) is a distributed knowledgebase for coordinating and researching Alzheimer Disease. By using semantic web concepts and variable privacy settings, researchers can collect information and collaborate while also avoiding duplicated effort. While SWAN handles data management, analytics activities must be carried out using other software packages.

3.3 | Graph Partitioning Strategies

Real-world graphs tend to be voluminous, requiring vertices and edges to be distributed over a cluster of machines. One unique aspect of dealing with distributed graphs is the impact of the partitioning strategy on load balancing and computational throughput; many networks exhibit connections with a power law distribution and cannot be simply divided uniformly across entities in the graph. Partitions often lead to load balancing problems, and synchronization between these partitions leads to high communication latencies. Optimal graph partitions reduce communication costs while also providing a uniform distribution of load. Therefore, the research community has concentrated on real-time graph partitioning schemes and partitioning without knowledge of the complete graph.

PowerGraph, a distributed graph processing framework, introduced the *oblivious* graph partitioning scheme (49). This strategy requires information regarding previous edge assignments to improve current assignments, but does not collect information from other partitions to avoid communication overheads. Sajjad et al. proposes the HoVerCut methodology, which can be coupled with existing graph partitioning algorithms (50). This method uses a distributed multithreaded environment where each thread is *subpartitioner* that executes the partitioning algorithm. Additionally, HoVerCut employs a windowing technique for batch updates to reduce the impact of shared state. This method is said to be the foundation of optimal communication between partitions. PowerGraph also introduced the PDS (Perfect Difference Set) scheme (51), which requires $(p^2 + p + 1)$ partitions where p is a prime number. Due to this constraint, PDS is used less frequently in real-world applications.

Several graph partitioning algorithms place emphasis on the *degree* of the vertex to employ better heuristics (52, 53, 54). These algorithms assign vertices to partitions based on a random hash function and then attempt to determine edge partitions. Xie et al. suggests a degree-based hashing algorithm, Libra (53), 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. proposes a stream-based, vertex-cut partitioning algorithm, High-Degree Replicated First

(HDRF) (52). This algorithm attempts to replicate high-degree vertices and maintains locality for low-degree vertices. Chen et al. employs the Hyrid-cut (54), 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. All edges are assigned to their respective target vertex partitions, followed by reassignment of edges with high-degree vertices (identified by a degree threshold). These algorithms produce better partitions for power-law graphs in terms of communication and load balancing since they replicate high-degree vertices. However, determining the degree of the vertices beforehand increases graph ingestion time.

4 | METHODOLOGY: SUPER-SPREADER IDENTIFICATION, CLASSIFICATION, AND ANALYSIS

Our goal is to identify and classify highly influential herds in the disease outbreak network. To achieve this goal, we have composed a workflow that comprises multiple analysis phases. As depicted in Figure 1 , there are 3 major phases. In Phase 1, we perform data preprocessing to extract features and create the disease transmission network that is leveraged by subsequent analysis steps. Phase 2 generates global herd rankings and influence measures from the DTN. Phase 3 focuses on characterizing highly influential herds by studying their epidemic attributes and modeling the relationship between the characteristics. We perform validation and evaluation for each phase in Section 4.4.

4.1 | Creating DTNs

NAADSM generates one output file per scenario. Results for each iteration are assembled based on simulation time steps. A data fragment from an iteration contains over 2000 input variables and 10-20 output variables, including the outbreak duration, number of infected herds, and vaccinations used.

Since scanning the raw data for each analysis step is not efficient at this scale, we removed initially infected herds from each of 3.2 million iterations. With the remainder of the dataset, we generated a weighted directed graph called the *disease transmission network* (DTN). The DTN is denoted as $G = (V, E)$, where V is the set of vertices, representing herds, and E is the set of edges, representing infection propagation. To create the DTN, we extract *infection propagation pairs* from the dataset, which are tuples that include the infected herd and source of infection. We use the Spark framework to compute infection percentages for every infection propagation pair, which are used as the weights for directed edges in the graph. For example, if A and B are two vertices connected by an edge with weight $1/5$, then A is source of infection in 1 out of 5 instances where B is infected. Apart from removing initially infected herds, we did not perform additional pruning on the DTN because our methodology is robust to noise from low-impact herds in the source dataset.

4.2 | Identifying Highly Influential Herds

Influential herds play a pivotal role in transmitting disease to their neighbors by making outbreaks last longer or become more severe. In these situations, the influence of a herd depends on the influence of its neighbors. In other words, a herd has high influence if it is infecting other highly influential herds. This type of interaction can be efficiently modeled by the PageRank algorithm.

4.2.1 | PageRank Algorithm

PageRank was proposed by Larry Page et al. (13) 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, which describes the probability that a random surfer (randomly clicking on links) will arrive at the web page. The higher the PageRank value, more important the web page is. In general, highly linked pages are more important than pages with a low number of incoming links. Further, 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 are from highly linked web pages, then 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. This notion of importance is similar to being influential; considerable research has been conducted on using PageRank to determine influence (20, 21).

4.2.2 | Using PageRank to Measure the Degree of Influence

Construction of the DTN produces a weighted, directed graph, where the weight of each edge is the rate at which one herd is infected by another. As a result, the sum of input links' weights must be equal to 1. When a disease is transmitted from vertex A to vertex B, we model the interaction as A *influencing* B. Similarly, vertex A influences all of its downstream neighbors. However, the PageRank algorithm computes the importance of herds based on *input* links, whereas in our case the influence of a vertex is decided by output links. Therefore, we invert the direction of edges 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. A demonstration of an inverted graph is provided in Figure 2 .

4.3 | Classifying Highly Influential Herds

After discovering influential herds, we provide two types of classifications to understand their characteristics. First, we classify the herds based on their likelihood to be super-spreaders. Second, we perform *localized* classifications to detect herds that have a particularly strong influence on another herd but not necessarily the system as a whole.

In epidemiology, the presence of super-spreaders is a phenomenon that is widely observed in disease outbreaks. A super-spreader is an infected herd that spreads the disease disproportionally to other herds

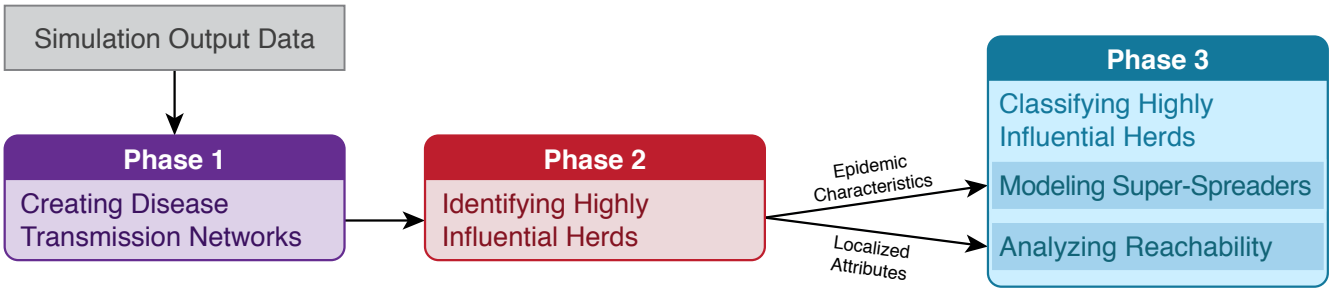

FIGURE 1 High-level overview of our analysis workflow.

(55). For a given outbreak, there may exist more than one super-spreader and the majority of individuals infect multiple secondary contacts. The most recent SARS outbreak involved super-spreading events (SSE) (56). In this section, we investigate classifying super-spreaders from the group of highly influential herds. Classifying super-spreaders helps provide more efficient planning that controls contacts such as shipments or veterinarian visits.

4.3.1 | Empirical Classification of Super-Spreaders

Super-spreaders tend to follow the Pareto principle (57), also known as the 80-20 rule, where approximately 20% of infected individuals are responsible for 80% of causality (14). A herd is also considered to be a super-spreader if it is responsible for a significantly larger percentage of transmission (28). To detect super-spreaders, we measure the per-herd *infection contribution* ($cont_{herdID}$) for each scenario by calculating the percentage of total infections caused by each herd. Infection contributions are collected from each scenario, averaged, and then sorted. We apply the 80-20 rule to select the top 20% of herds in descending order as probable super-spreaders, with all herds of equal ranking in the top 20% considered. Using this methodology, we observed that the top **23.43%** infection contributors were responsible for **68.85%** of the infections. This result provided a foundation for attribute-based modeling and classification.

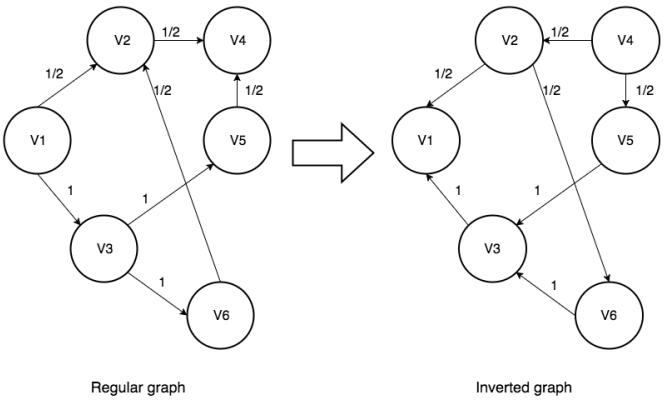

FIGURE 2 Formation of an inverted graph of disease transmissions for use with the PageRank algorithm.

4.3.2 | Model-Based Classification of Super-Spreaders

Super-spreaders behave differently from the rest of the population, but determining *why* a particular herd becomes a super-spreader can provide high-level insight for disease spread analysis. Potential features that often influence super-spreaders include (55):

- *Degree of local infections*: Number of herds directly infected by a particular herd
- *Depth of disease transmission*: Length of the traversal path through the disease transmission network due to the associated herd's infection
- *Rate of contribution*: Percentage of the total number of infected herds
- *Level of Infection*: Relative position of the herd in the infection chain hierarchy

We backtrace through the disease transmission network to determine each of these properties. After collecting training data for each herd across our subject dataset, we applied multiple machine learning classifiers: support vector machines (SVMs) (58), random forests (59), and quadratic discriminant analysis (QDA) (60). An initial exploration of these models' hyperparameters found that the classifications produced by SVMs exhibited the highest performance. To train the SVMs, we used *stochastic gradient descent* (SGD). SGD is a stochastic method for finding local minima or maxima by updating a set of parameters iteratively to minimize an objective function (61). The major advantage of SGD is its efficiency and amenability to parallel computation, which ensures scalability in our particular use case (62).

4.3.3 | Reachability Analysis via Localized Attributes

Up to this point, discussion has revolved around determining influential herds across the entire disease transmission network. However, there

are often localized relationships between herds that are significant but not highlighted by global analysis. Determining localized influence for a particular subset of herds is useful in situations where a planner wishes to isolate an infection or slow the spread of disease. These relationships are measured by the *localized* influence value, which is calculated based on Formula 1:

$$Infl\_val_{ij} = \frac{NPR_i * NOC_{ij}}{Avg\_dist_{ij}} \quad (1)$$

Where:

$Infl\_val_{ij}$ = Influence value of herd *i* on herd *j*

NPR_i = Normalized PageRank value (1-10) of herd *i*, representing global influence in the DTN

NOC_{ij} = Normalized occurrence count (1-10) of herd *i* when herd *j* is infected

Avg_dist_{ij} is a measure of distance between herd *i* and herd *j*, which is calculated by the following formula:

$$Avg\_dist_{ij} = \frac{\sum_{k=1}^{n} dist_k(i,j)}{n} \quad (2)$$

Where:

n = Number of times herd *i* is infecting herd *j*

$dist_k(i,j)$ = Distance between herd *i* and herd *j* in hops for k^{th} occurrence

This results in herds having more influence on those in close proximity. For instance, a herd that is a single hop away is more influential than a herd that is two hops away in the DTN. Dividing NPR_i by Avg_dist_{ij} gives an approximate value of influence of herd *i* on herd *j*. By using NOC_{ij}, we increase the importance of herds that are infected often by a another herd.

4.4 | Evaluation

Herein we evaluate our methodology; our benchmarks include machine learning classifications, statistical evaluations, and analysis of the scalability of our approach using Apache Spark.

4.4.1 | Classifying Super-Spreaders with Machine Learning

Using the DTN to backtrace through herd interactions, we generated training data based on features that commonly indicate super-spreaders (as described in section 4.3.2). Herd classifications were stored in this dataset as a binary value, with 1 indicating a super-spreader and 0 representing a regular herd. Our baseline classification via the 80-20 rule was used as ground truth, and we applied several machine learning algorithms on the training data. Classifications were implemented with scikit-learn (62), and a randomized 90-10 split was used for the training and testing datasets, respectively. As depicted in Table 1 , the SVM model provided the highest accuracy. However, it is worth noting that each of the machine learning algorithms achieved reasonable accuracy based on our feature set.

TABLE 1 Accuracy for each machine learning classification algorithm evaluated. To demonstrate generality, we also used our SVM model on a different scenario set in Iowa, USA.

Classifier	**Accuracy**
Quadratic Discriminant Analysis	83.97%
Random Forest Classifier	88.9%
Support Vector Machine (SVM)	**90.02%**
SVM, Iowa Dataset	93.50%

One of the primary benefits of generating machine learning models is generalizability; if the model generalizes well, then it can predict super-spreaders in new or unseen datasets without needing to perform analysis over the disease transmission network. To evaluate the generality of our SVM model trained on the Colorado dataset, we obtained a second scenario set in Iowa, USA, which consisted of 8 TB of simulation output. Using the model, we were able to predict super-spreaders with an accuracy of 93.50% as shown in Table 1 . This is likely due to some similarities in parameters between the two scenarios, as both simulated an outbreak of foot-and-mouth disease.

After the algorithms are fully trained, coefficients associated with the features capture their respective impacts on classification. We provide these coefficients as outputs during the modeling process. Coefficients from our SVM classifier are shown in Figure 3 ; positive weights suggest a positive correlation with the output (classification as a super-spreader or not), and vice versa. Based on these results, the *degree of local infections* exhibits a strong correlation with the herd in question being a super-spreader, which is also true of SARS outbreaks (30). Conversely, the level of infection in the DTN hierarchy was negatively correlated with being a super-spreader, and the contribution rate and depth of disease transmission were not weighted as highly for this particular model.

4.4.2 | Statistical Evaluation of Super-Spreaders

To understand the composition of highly influential herds, we applied a variety of statistical techniques on the data produced by our disease transmission network. Our analysis includes a proportion test, ROC (receiver operating characteristic) curves for the experiments, as well as a breakdown of seeders, super-spreaders, and combined influential herds.

4.4.3 | Highly Influential Herds vs Super-Spreaders

We performed a two-sample proportion test to statistically support our claim that herds with high PageRank values include higher proportions of super-spreaders. In this evaluation, we assessed the top 20% of PageRanked herds (likely super-spreaders) with the next 20% (referring back to the 80-20 rule). To conduct the proportion test, we randomly selected 1000 herds from each set and noted the mean number of

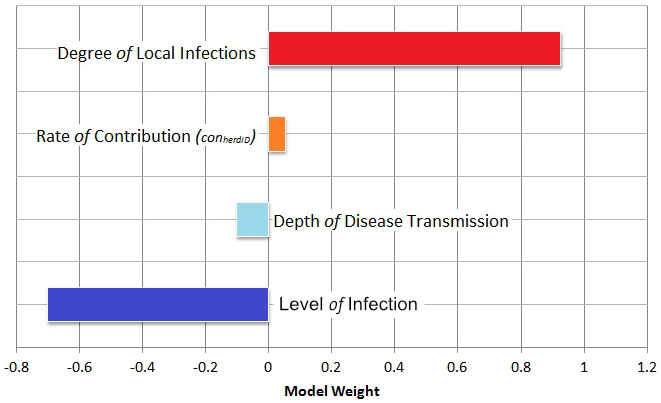

FIGURE 3 Feature coefficients from our SVM classifier; larger values indicate more influential features.

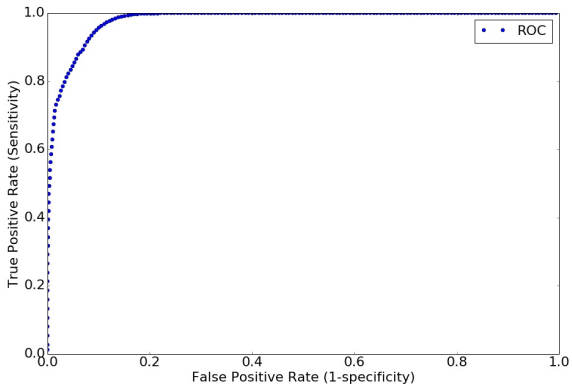

FIGURE 4 ROC curve for herds classified as super-spreaders compared with herds that exhibited high PageRank values.

super-spreaders across 40 iterations: $\bar{x}_{top} = 839.93, \bar{x}_{next} = 192.5$. This experiment revealed a significant difference between high PageRank ($\hat{\pi}_{top} = 0.83993, n_{top} = 1000$) and low PageRank ($\hat{\pi}_{next} = 0.1925, n_{next} = 1000$) herds; $z* = 8.657, p < 0.00001$ for $\pi_0 = 0.5$ with a 95% confidence interval $[0.614, 0.6808]$. These results suggest that herds with high PageRank values contain a higher proportion of super-spreaders compared to the population with low PageRank values.

In the next part of this experiment, we analyzed the inclusion of super-spreaders in the composition of highly influential herds. We found 3747 probable super-spreaders using the approach described in section 4.3.1. We then calculated the number of herds having the top *n* PageRank values among the 3747 super-spreaders, $n \in \{50, 100, 200, \cdots, 18800\}$. The ROC curve for this experiment is shown in Figure 4 . Based on the curve, the experiment resulted in high accuracy, meaning super-spreaders account for a considerably large portion of the overall set of influential herds. The reason behind this result is that both groups infect a higher number of herds on average; according to Figure 3 , 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 4.2.1. Moreover, we can observe that the likelihood ratio is decreasing as we move along 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 its counterpart.

4.4.4 | Highly Influential Herds vs Seeders

This experiment analyzes the involvement of seeder herds (herds that are infected by the set of initially infected herds) in the evolution of super-spreaders. As described in Section 4.1, we remove initially infected herds from the infection propagation pairs and collect the rest of the data for analysis. Over the 3.2 million iterations, we found 6504 distinct seeders. We performed same experiment as described in the previous section (4.4.3), except this time the number of herds having the top *n* PageRank value are among 6504 seeders instead of super-spreaders, $n \in \{50, 100, 200, \cdots, 18800\}$. The ROC curve for this experiment is shown in Figure 5 ; we can observe a small peak initially, followed by monotonic increases afterwards. The area under the curve is much less compared with 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 very few 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 a little contribution towards their own influence.

The True Positive Rate (TPR) and False Positive Rate (FPR) used to create the ROC curves in the previous experiments are calculated using following formula:

$$TPR_n = \frac{NI_n}{T_p}$$
$$FPR_n = \frac{n - NI_n}{T_n} \quad (3)$$

Where:

NI_n = Intersection of super-spreaders or seeders with the top n highly influential herds

T_p = Total number of super-spreaders or seeders

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

4.4.5 | Scalability Evaluation

We measured the time taken by the Spark framework to compute PageRank values of herds in the disease transmission network for various combinations of data and cluster sizes. From the 100,000 simulation outputs in our Colorado dataset, we extracted disease transmission information in the form of infection propagation pairs and executed our PageRank implementation. We considered cluster sizes with a varying number of machines, each of which was accountable for four Spark

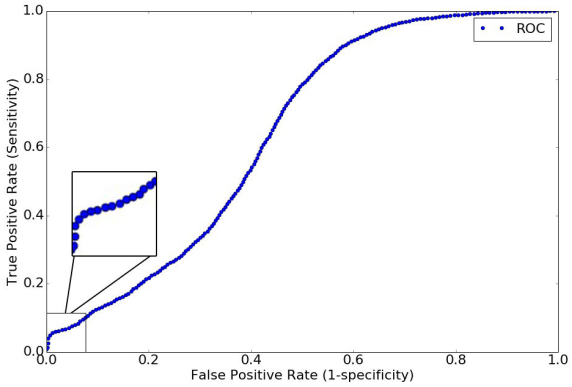

FIGURE 5 ROC curve for herds classified as seeders compared with herds that exhibited high PageRank values.

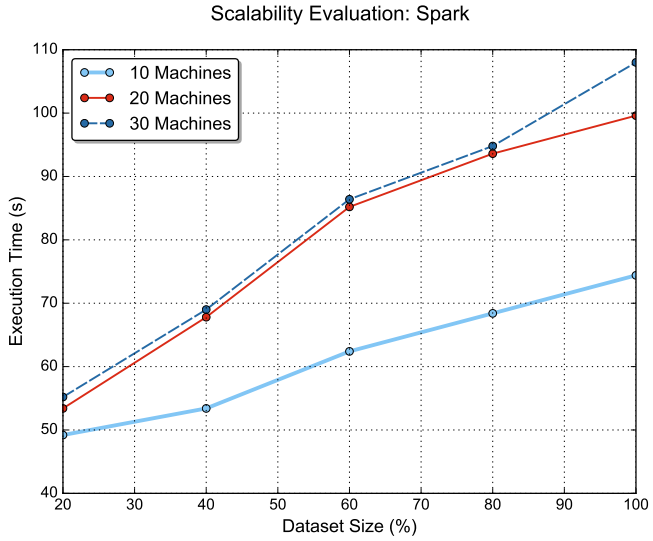

FIGURE 6 Scalability of our approach executing under the Apache Spark framework. By increasing the cluster size to 30 machines, we reduce the execution time by about 25%.

workers. Figure 6 demonstrates the results of this benchmark; the vertical axis contains the time taken to perform the computation, with dataset sizes 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.

5 | METHODOLOGY: GRAPH PARTITIONING

A single commodity machine completes a typical NAADSM simulation run in about 20 hours (16, 63). Subsequent analysis of the DTN is an iterative process that can far exceed simulation execution time. To facilitate faster and more efficient analysis, we adapted our disease transmission network to run within the Apache Spark GraphX framework (12). Given that PageRank is a graph algorithm (13), GraphX is well-suited for our purposes.

In an ideal scenario, the input graph representing our DTN will be partitioned uniformly across a cluster of machines. In practice, however, super-spreaders have a disproportionate effect on graph characteristics and lead to imbalances in load. These imbalances increase both network communication and graph ingestion time. To improve performance, we extended the underlying 2D graph partitioning scheme used in GraphX. This enhanced partitioning scheme ensures that each vertex in the graph saves up to two network transfers compared to the standard partitioner while distributing a nearly uniform amount of load among machines, allowing the PageRank algorithm that underpins our analysis to complete faster.

5.1 | Graph Partitioning Background

Graphs and their related algorithms are applied in a wide variety of domains, including social networks, recommendation systems, web documents, routing, and more. These applications have grown considerably over time; the Facebook friend graph has more than 1 billion vertices (users) and nearly 1 trillion edges (friendships) (64). Moreover, many machine learning techniques (such as deep neural networks) can be represented as graphs. Applications in these domains demand nearly real-time results. Although computing and storage capacities continue to increase, a single machine is unable to satisfy these real-time requirements completely. This led to the advent of distributed graph computation systems (also called distributed graph-parallel systems), which operate on clusters of commodity hardware (65, 66, 67, 12). These systems employ the "Think Like a Vertex" (68) programming model and expose developer-friendly APIs wherein any graph algorithm can be expressed from the perspective of a vertex. The partitioning algorithm used with these graphs directly impacts performance in a distributed setting, influenced by the following:

- *Ingestion time*: this refers to the time taken by the system to partition and load the graph before starting analysis. Some partitioning strategies incur more ingestion time than others.
- *Communication costs*: the amount of network transfer between partitions. Since partitions are stored on different machines, some amount of communication is required to achieve synchronization. This plays a prominent role in the performance of the partitioning scheme as well as 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. Improper distribution leads to under- or over-utilization of cluster resources.

Balanced graph partitioning is classified as an NP-complete problem (69), meaning that solutions are generally acquired by heuristics or approximation. With the arrival of graph-parallel systems, the research community has taken interest in streaming graph partitioning algorithms, which partition a vertex or an edge based only on local information rather than the complete graph. These algorithms are divided into two categories: *edge-cut* and *vertex-cut*.

5.1.1 | Edge-Cut

Edge-cut algorithms allocate vertices in a roughly uniform fashion across partitions, and then cut and distribute the edges across partitions to produce complete subgraphs. It is worth noting that cut edges and their corresponding vertices are replicated. Due to this characteristic, the communication cost associated with edge-cut algorithms is directly proportional to the number of edges cut. Each partition is responsible for an approximately equal number of vertices, whereas the number of edges is variable and determines the load-balancing factor.

Figure 7 (a) shows an example of edge-cut partitioning on a graph with 5 vertices and 11 edges. The goal is to divide the graph into two parts, with each part representing a complete subgraph. As mentioned previously, vertices are partitioned first. In this example, they are partitioned using an elementary hash partitioner:

$$v \bmod n$$

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. Given the hypothetical cut shown in Figure 7 (a), the destination partitions of edges $(1,2)$, $(5,2)$, $(5,4)$, $(2,5)$, $(4,5)$, and $(4,3)$ are ambiguous. To place these edges, a deterministic scheme must be employed; suppose the edges are placed into their destination vertex's partition, meaning edge $(1,2)$ would be placed in the same partition as vertex 2. This scheme would produce incomplete subgraphs; for example, all of the edges of vertex 2 are not in the same partition — edge $(2,5)$ is in partition 1. In order to function properly, these subgraphs need to be complete. Therefore, some of the vertices and edges are replicated (denoted by red dashed lines in Figure 7 -b), leading to increased communication.

5.1.2 | Vertex-Cut

In vertex-cut algorithms, edges are distributed equally among partitions and vertices are cut and replicated to produce complete subgraphs. In this scheme, communication costs are directly proportional to the number of vertex replicas, while the load-balancing factor is determined by the number of edges assigned to each partition.

Figure 8 shows vertex-cut partitioning. As in the previous example, a directed graph with 5 vertices and 11 edges is provided. Edges are distributed equally among the partitions using a function such as:

$$(V_s + V_d) \bmod n$$

Where V_s is the source vertex and V_d is the destination vertex. This places edges $(1,2)$, $(5,2)$, $(2,5)$, $(5,4)$, $(4,5)$, $(4,3)$ in partition 1, with the rest placed in partition 2. Furthermore, suppose vertices are partitioned using our elementary hash partitioner, $(v \bmod n)$. Like the previous example, this strategy also creates incomplete subgraphs. However, this strategy only transmits vertex data between partitions (represented by vertices highlighted in red).

5.1.3 | Evaluating Partition Quality

To evaluate partition quality, we use the following criteria:

- *Replication Factor*: amount of replicated vertices/edges. This measure is directly proportional to the communication between partitions.
- *Load Balancing Factor*: distribution of load among partitions, represented by the standard deviation of the number of edges for which each partition is responsible. A lower load balancing factor indicates better utilization of cluster resources.

An ideal scheme achieves uniform distribution while incurring minimal replication. Ingestion time is often directly proportional to the quality of partitions, with longer ingestion times producing better partitions.

Early graph-parallel systems, such as Pregel (65) and GraphLab (67), employed the edge-cut partitioning scheme. Subsequent research discovered significant load balancing issues in the case of power-law graphs (49) (also known as real-world graphs or natural graphs), due to large edge imbalances. Therefore, Powergraph (49) employed vertex-cut partitioning and reported a significant improvement in power-law graph partition quality. Successive graph-parallel systems including Spark GraphX (12) have also employed some form of vertex-cut partitioning. 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 since edges are evenly distributed.

5.2 | 2D Graph Partitioning

Modern graph-parallel systems employ vertex-cut partitioning guided by a variety of heuristics. 2D partitioning (also known as grid-based partitioning) was proposed by Nilesh et al. (70) and claims an upper bound of $2\sqrt{n} - 1$ on the vertex replication factor, where n is the number of partitions.

Graphs possess two properties: vertices and edges. The identifier space of vertices is one-dimensional because we can assume each vertex has a unique ID. However, the identifier space of edges is

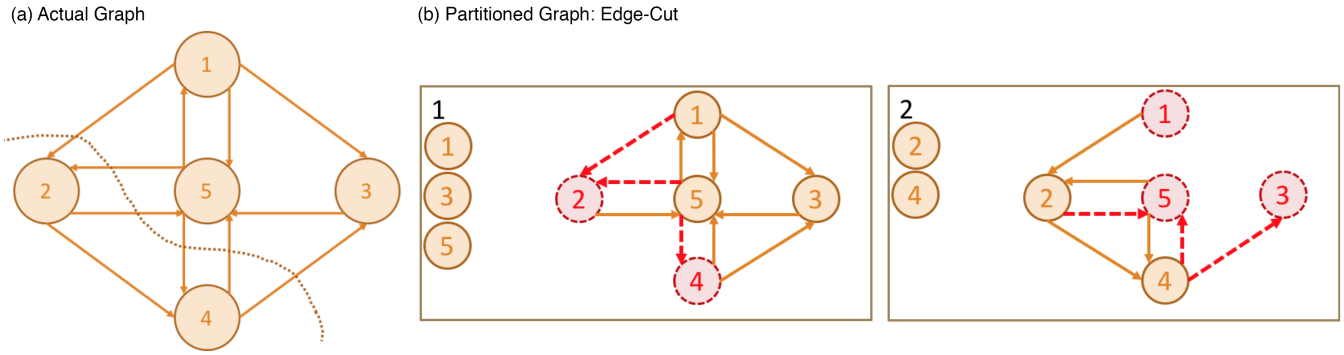

FIGURE 7 A demonstration of edge-cut partitioning.

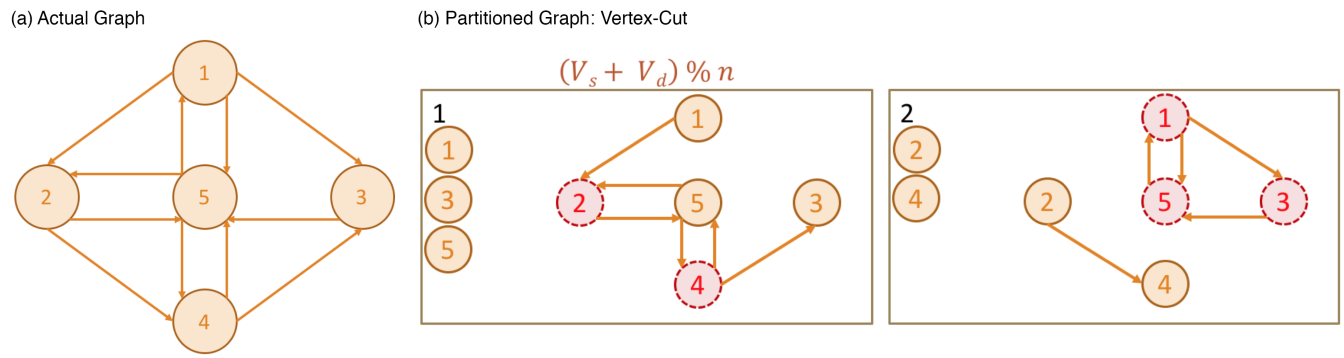

FIGURE 8 A demonstration of vertex-cut partitioning.

two-dimensional because source and destination vertex identifiers are required to uniquely identify an edge. Due to this requirement, a grid can be leveraged to hold edge properties. In this strategy, partitions are assumed to be a grid of rows and columns. The vertex identifier space is mapped to this grid using hash functions, with the intersections of the rows and columns representing edges.

5.2.1 | EdgePartition2D

The 2D partitioning strategy used in GraphX is called EdgePartition2D (71). Consider a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Every vertex V has a unique identifier, and every edge E contains a source and destination vertex. The goal here is to divide the graph into n parts such that the resulting partitions incur minimum communication with near-uniform distribution. Herein we explain this partitioning strategy in detail; the complete process is pictured in Figure 9 .

Logically, the partition space in EdgePartition2D is a two-dimensional grid. If n is a perfect square, then the grid will have an equal number of rows and columns (hereafter referred to as $rows$ and $cols$, respectively), or $\sqrt{n}$. On the other hand, if the grid is not square, $cols$ can be at most one greater than $rows$. In this situation, $cols$ is the ceiling

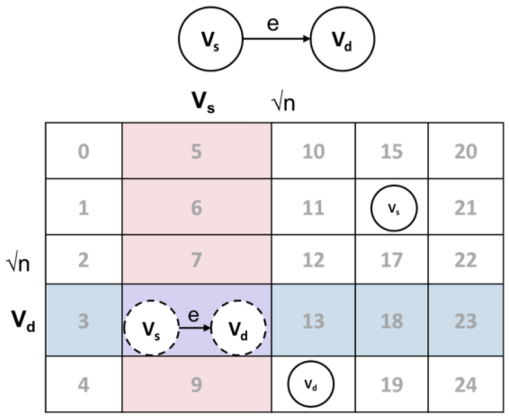

FIGURE 9 Vertex and edge placement in the EdgePartition2D algorithm.

of the decimal value of $\sqrt{n}$ and $rows$ is expressed as:

$$rows = \frac{n + cols - 1}{cols}$$

Every column has this count of 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.

Vertices are assigned to partitions based on the elementary modular hashing discussed previously. Consequently, vertices are equally distributed among partitions. Edges, on the other hand, are assigned by hashing both the source and the destination vertices. The source vertex identifier space is mapped to 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 by:

$$col = (src * mixingPrime) \bmod \sqrt{n}$$

If not, col is identified by:

$$col = \frac{(src * mixingPrime) \bmod n}{rows}$$

Where $mixingPrime$ is a large prime number that is used to improve the balance of edge distributions (71).

Similarly, destination vertices are mapped to rows. Hence, the row selection for an edge is accomplished by the destination vertex, hereafter denoted by dst. If n is a perfect square, the row is identified using:

$$row = (dst * mixingPrime) \bmod \sqrt{n}$$

Otherwise:

When $col < cols - 1$:

$$row = (dst * mixingPrime) \bmod rows$$

When $col \geq cols - 1$:

$$row = (dst * mixingPrime) \bmod lastColRows$$

From the perspective of a vertex, the EdgePartition2D scheme helps us determine the upper bound on the vertex replication factor. Consider a vertex with identifier v which could be either a source or destination in the graph. All edges where v is the source vertex would be placed in the same column, col, of the grid. Similarly, the edges where v is the destination vertex would be placed in the same row. Therefore, 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, shown in Figure 10 .

5.3 | Enhanced 2D Partitioning

EdgePartition2D excels at load balancing while also reducing communication between partitions. However, the vertex partitioning factor in EdgePartition2D is not completely optimized; as mentioned previously, 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 v is the destination vertex are placed in partitions of the j^{th} row, $j <= r$. Given this, v would be ideally placed in the intersecting partition of the i^{th} column and j^{th} row. Instead, it is placed in the partition expressed by $v \bmod (r * c)$.

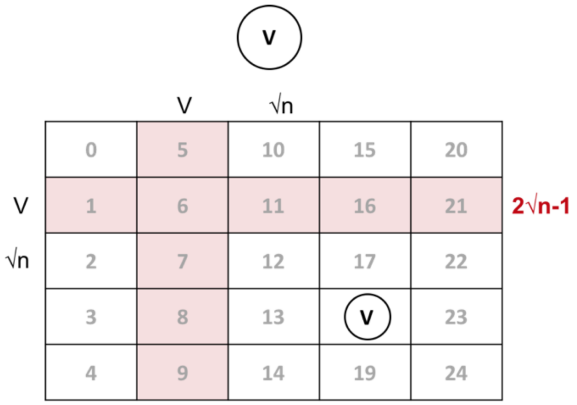

FIGURE 10 Upper bound on the replication factor of the EdgePartition2D algorithm.

An improved version of this algorithm places v in the partition where a hypothetical edge with v as both the source and the destination vertex would be placed. However, this modification raises load balancing issues; consider n partitions where n is a perfect square resulting in a grid with $\sqrt{n}$ rows and $\sqrt{n}$ columns. As described previously, a square grid would employ the same hashing scheme for both columns and rows, $(v * mixingPrime) \bmod \sqrt{n}$. Applying this vertex partitioner would result in all vertices being placed in the diagonal partitions of the grid. In other words, the non-diagonal partitions would not accommodate any vertices. However, it is possible to achieve a better balance of load by employing different hashing techniques for source and destination vertices.

Given a perfect square for the number of partitions, n, our approach selects the column for vertex v using the following hash function:

$$\frac{(v * mixingPrime) \bmod n}{\sqrt{n}}$$

While row selection remains the same as the default implementation. Since the modified column selection leverages both the modulo and the division operation, the column index differs from the row index. This enables vertices to disperse over the grid. For instance, consider:

$$n = 100$$
$$v = 1546$$
$$mixingPrime = 1$$

With the standard EdgePartitioner2D, v would be placed into the $(1546 * 1) \bmod 10 = 6^{th}$ column, and $(1546 * 1) \bmod 10 = 6^{th}$ row in the grid of 10×10. With our approach, it would be placed into $((int)((1546 * 1) \bmod 100/10)) = 4^{th}$ column and 6^{th} row.

Due to the vertex being 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 $2\sqrt{n} - 2$, or one less than the existing implementation, illustrated in Figure

11 . For an iterative algorithm like PageRank, this avoids two network transfers per vertex per iteration, improving overall performance.

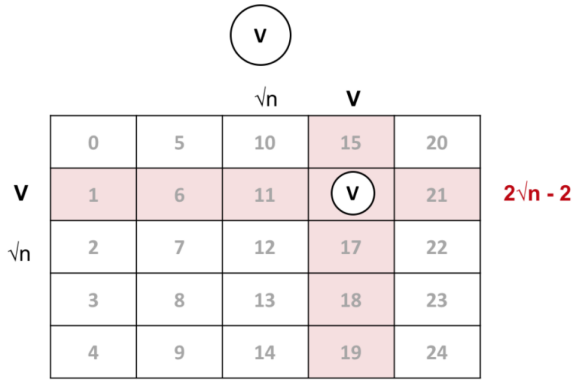

FIGURE 11 Replication factor of our enhanced partitioning approach.

Given the difference in hashing techniques employed for source and destination vertices, distribution throughout the grid is balanced even in the case of a perfect square for the number of partitions. Although our modified partitioning scheme changes the source vertex hashing function, the destination hash remains the same. Therefore, the partitions identified by the EdgePartition2D scheme and our approach would differ for a particular edge, but load balancing properties are nearly preserved.

5.4 | Evaluation

Herein we evaluate our enhanced 2D graph partitioning strategy in Spark GraphX. We compare our approach with EdgePartition2D in terms of graph ingestion time, vertex replication factor, PageRank execution time, and workload distribution. All of the experiments were conducted on the inverted graph (DTN) generated from our subject dataset, which comprised 18,890 vertices and 1,682,361 edges. Results were collected for a variable number of partitions: {10, 25, 50, 75, 100, 125, 144, 200}. This set contains both non-square and square numbers to embody all aspects of the candidate algorithms.

5.4.1 | Scalability

To evaluate the performance of our algorithms under Spark GraphX, we re-ran the scalability benchmark outlined in Section 4.4.5 using the GraphX-based implementation. The experiment measured the time taken by the framework to compute PageRank values of herds in the disease transmission network for various combinations of data and cluster sizes. 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 accounted for four

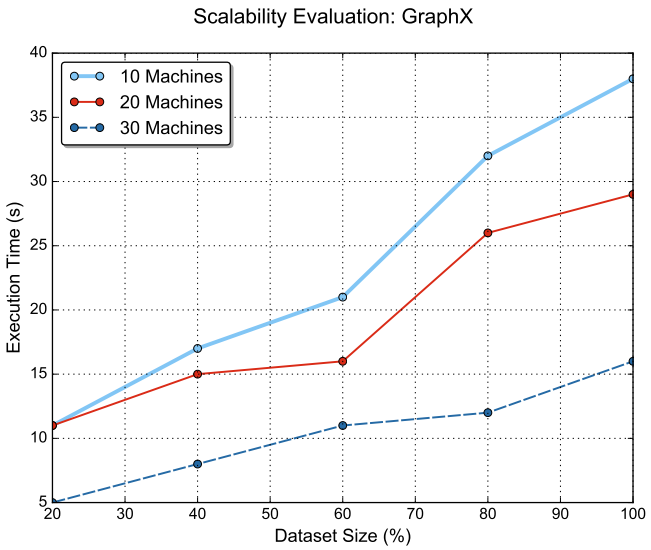

FIGURE 12 Evaluation of the scalability of our approach using GraphX with variable cluster and dataset sizes.

Spark workers. Figure 12 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.

Results are similar to those in the previous experiment, but the PageRank process completes much faster under GraphX (roughly 16 seconds versus 73 seconds in the previous implementation with 100% of the dataset and 30 machines). Figure 13 illustrates the execution speed improvements in the GraphX version of the DTN over our previous Spark implementation. The Resilient Distributed Graph (RDG) structure in GraphX is designed specifically for our particular use case, which accelerates graph-specific processing (12).

5.4.2 | Graph Ingestion Time

As discussed previously, graph ingestion time refers to how long it takes to partition and load the graph into memory. Conceptually, this is the time taken by the partitioning algorithm starting from dividing the graph and creating subgraphs up to loading each element into memory for further computation. Partitioning may affect the ingestion time adversely, so it is worth studying and comparing with the previous implementation. Figure 14 shows the comparison of ingestion 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.

5.4.3 | Replication Factor

The vertex replication factor describes the amount of replication incurred by the partitioning algorithm and is directly associated with

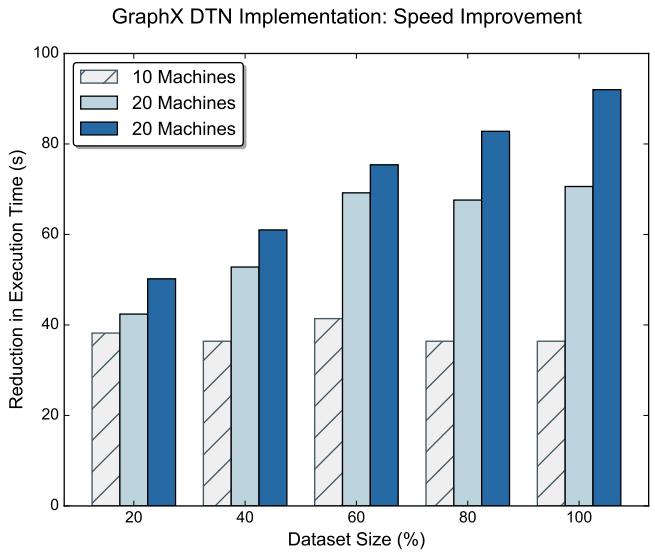

FIGURE 13 A comparison of the GraphX version of our DTN with the previous Spark implementation. The reduction in execution time is shown for each dataset and cluster size.

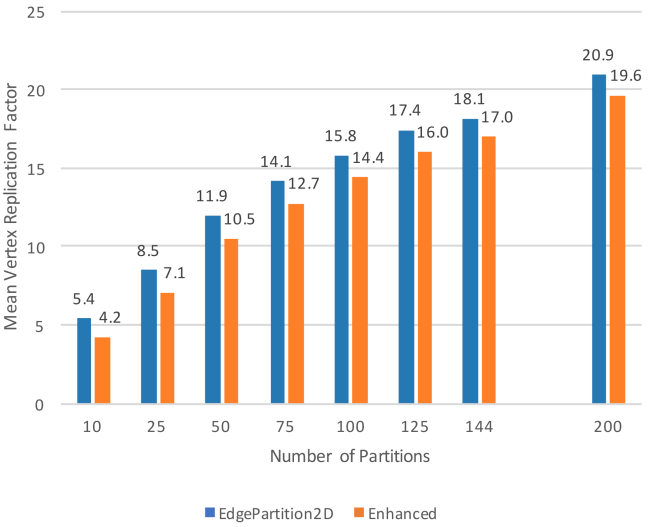

FIGURE 15 Comparison of vertex replication factors between EdgePartition2D and our approach. In all configurations our enhanced approach reduces the replication factor (by 1.3 on average).

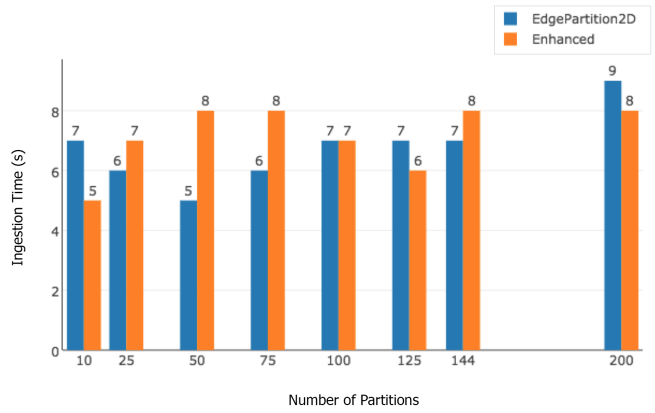

FIGURE 14 Comparison of graph ingestion time. Our enhanced approach does not introduce a substantial change in ingestion performance over EdgePartition2D.

the amount of communication required. Overall, the replication factor is the key metric that decides the performance of any partitioning algorithm. In other words, a smaller replication factor implies less communication and better performance. Figure 15 shows a comparison between the original and enhanced partitioning algorithm in terms of the mean vertex replication factor for a variable set of partitions. In this benchmark, our enhanced partitioning strategy demonstrated better replication factor results than the standard EdgePartition2D scheme. It is worth noting that the difference in replication factor for any particular observation is more than one. The reason behind this behavior is that our algorithm decreases the upper bound on vertex replication from $2\sqrt{n}-1$ to $2\sqrt{n}-2$.

In order to determine whether the differences between mean replication factors was statistically significant, we performed the Wilcoxon signed-rank test because the distribution was not normal ($p-value = 0.03744$ for Shapiro-Wilk normality test). The Wilcoxon signed-rank test demonstrated that the difference between mean replication factor of both the approaches is 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 our enhanced algorithm is at least one less than that in EdgePartition2D.

5.4.4 | Load Balancing: Vertex Distribution

Load Balancing refers to the distribution of graph elements (vertices and edges) across partitions. A key issue that affects this metric is that workload distribution and the replication factor are conflicting interests, with improvements to one often having a negative impact on 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, the strategy delivers unbalanced partitions in terms of edge distribution. In contrast, an ideal scheme keeps both of these factors within an acceptable bound. We measure the balance of load with side-by-side boxplots of vertex distributions, shown in Figures 16 and 17 . The observed similarity between the approaches along with the reduction in upper bound of the vertex replication factor is the prime reason for improvement in the performance of our enhanced approach. Figure 16 depicts the comparison in the vertices distribution for $n = 75$, and Figure 17 demonstrates the variation in distributions for $n = 125$ and $n = 144$. These results demonstrate that our enhancements to EdgePartition2D do not have a negative impact on load balancing. It can be observed in Figure 17 that vertices are distributed in a balanced

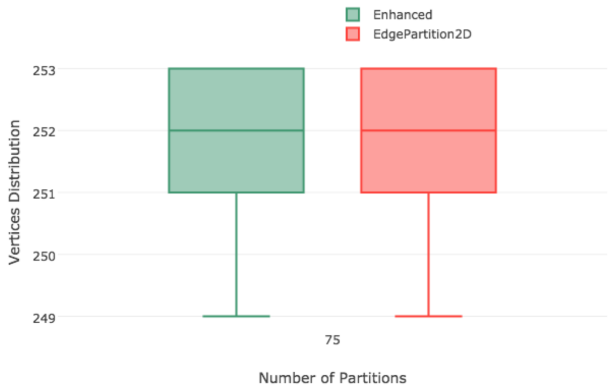

FIGURE 16 Comparison of vertex distributions, $n = 75$.

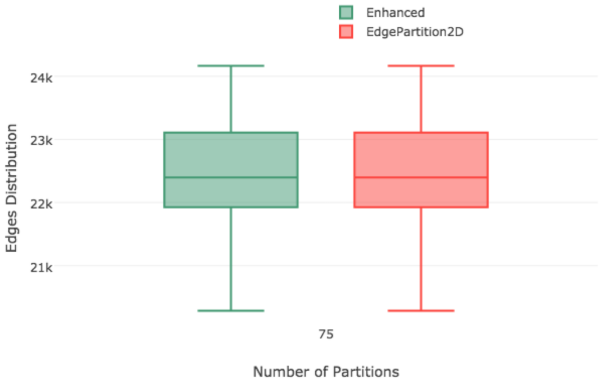

FIGURE 18 Comparison of edge distributions, $n = 75$.

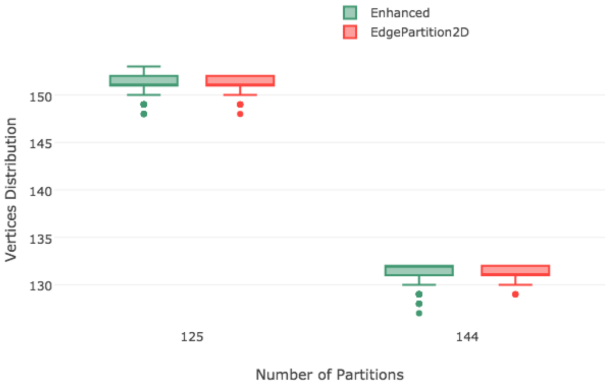

FIGURE 17 Comparison of vertex distributions, $n = 125$ and $n = 144$.

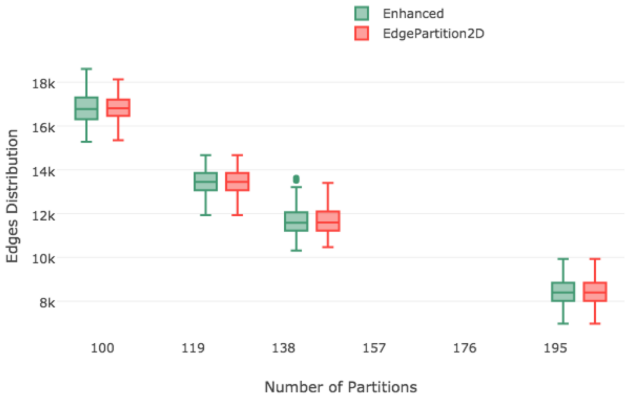

FIGURE 19 Comparison of edge distributions, $n = \{100, 125, 144, 200\}$.

fashion even with a perfect square for the number of partitions due to the algorithm employing different hashing techniques for the source and the destination vertices.

5.4.5 | Load Balancing: Edge Distribution

Figures 18 and 19 display the edge distribution across a variety of partition sizes, including both perfect and imperfect squares. The distributions remain similar for both algorithms, which is to be expected as the edge partitioning logic for imperfect squares is untouched by the new algorithm. Additionally, because only the source vertex hash has changed, the row assignments will remain the same.

5.4.6 | Execution Time

In this benchmark suite, *execution time* refers to the amount of time consumed by the algorithm starting from the creation of the graph to the completion of the algorithm. This metric is affected by all three of the previous metrics discussed in this section. Figure 20 shows the execution times of both algorithms. We can observe that our algorithm consumed less or an equal amount of execution time compared to EdegePartition2D. The primary reason for this is the reduction in mean vertex replication factor allows the algorithm to reduce communication and complete the partitioning process faster.

A Paired Samples t-test was performed to compare EdgePartition2D to our enhanced algorithm in terms of execution time. We found a significant difference in execution time between the enhanced version ($M = 36.75, SD = 17.04$) and EdgePartition2D ($M = 39.5, SD = 15.56$); $t(7) = 3.671, p < 0.005$.

6 | CONCLUSIONS AND FUTURE WORK

In this study, we presented our methodology for identifying epidemiologically influential herds and understanding their characteristics over voluminous data. Identification of influential herds will help planners allocate limited resources more effectively. Our methodology includes multiple analysis components such as: (1) generating a disease network data structure, (2) estimating the influence of a particular herd using the PageRank algorithm, and (3) characterizing influential herds based on their epidemiological characteristics and herd-based relevance.

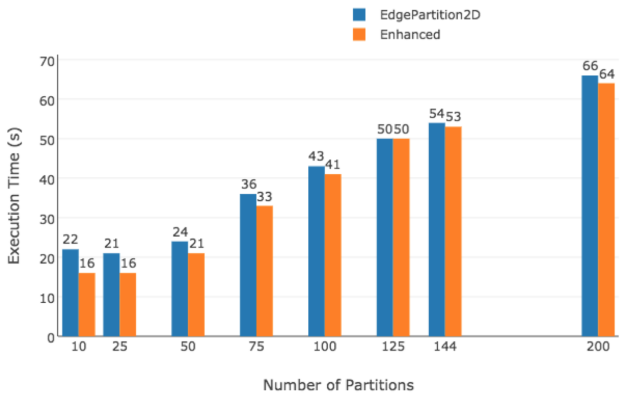

FIGURE 20 Comparison of execution times between EdgePartition2D and our enhanced version of the algorithm, in seconds.

RQ1 What data structure(s) allow us to represent disease spread interactions for analysis?

To achieve effective analysis with reasonable latency, we extract entire chains of infections from the output dataset and construct a graph-based disease transmission network (DTN) 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.

RQ2 How can we measure the influence of each herd?

We leverage the PageRank algorithm to estimate the influence of each herd in the DTN. The PageRank associated with a herd represents the probability that it contributes to a random infection chain. Our statistical analysis demonstrates that super-spreaders are well-represented among the highly influential herds. We have modeled the relationship between features of a herd extracted from the DTN and the likelihood of being a super-spreader using support vector machines (SVMs). Our model provides an accuracy of greater than 90% for FMD outbreaks in the state of Colorado; furthermore, this model transfers well and has an accuracy of over 93% when analyzing likely outbreaks in Iowa. This result demonstrates the generalizability of our methodology.

RQ3 How can we enable the analysis at scale?

Our analysis and experiments were performed using Apache Spark and were distributed across a cluster of computing resources. This approach was shown to be effective and scalable in our benchmark evaluation.

RQ4 Given a data model for disease transmission, how can we improve performance in a distributed setting?

We extended the graph partitioning algorithm in Spark GraphX, EdgePartitioner2D, to reduce network communication and speed up analysis. To achieve this reduction in communication, our modifications to the algorithm trade off a small amount of load balancing uniformity. We believe that this approach is broadly applicable, and best suited for applications that require frequent graph construction or communication between nodes. This is especially important in use cases with limited network bandwidth, such as Internet of Things (IoT) deployments where distributed nodes may also participate in taking measurements, collecting observations, and then building the DTN.

Our approach facilitates planning for disease outbreaks by pinpointing sources of infection that have significant contributions to disease spread. Our system architecture ensures such analysis is fast, efficient, and can make use of distributed resources at scale.

6.1 | Future Work

As part of our future work we plan to explore the feature space to improve the accuracy of our super-spreader detection model. We will extend the DTN data structure to include other features such as herd types, time-series data, and quality measures. Another avenue for future research is to leverage input parameters that are used for simulation variants to model the relationship between input features and highly influential herds. Moreover, the underlying graph-parallel system can be further extended to employ a partitioning strategy specifically designed for the updated version of the DTN. We may also experiment with other types of models/simulations and compare the results of real-world datasets with simulated data, if available, with the goal of eventually modeling live outbreaks. Finally, we plan to implement support for an incremental PageRank algorithm that can accommodate changes in the DTN in real time.

ACKNOWLEDGEMENTS

This work was supported by the US Department of Homeland Security (D15PC00279) and the US National Science Foundation's Advanced Cyberinfrastructure and Computer Systems Research Programs (ACI-1553685, CNS-1253908).

References

[1] Brooks-Pollock Ellen, Jong MCM, Keeling Matthew James, Klinkenberg Don, Wood James LN. Eight challenges in modelling infectious livestock diseases. *Epidemics*. 2015;10:1–5.

[2] Keeling Matt J, Rohani Pejman. *Modeling infectious diseases in humans and animals*. Princeton University Press; 2008.

[3] Harvey Neil, Reeves Aaron, Schoenbaum Mark A, others . The North American Animal Disease Spread Model: A simulation model to assist decision making in evaluating animal disease incursions. *Preventive veterinary medicine*. 2007;82(3):176–197.

[4] Pendell D.L., Leatherman J., Schroeder T.C., Alward G.S.. The economic impacts of a foot-and-mouth disease outbreak: a

regional analysis. *Journal of Agricultural and Applied Economics*. 2007;39(0):19–33.

[5] Green C., Whiting T., Duizer G., et al. Simulation modeling of alternative control strategies for an HPAI outbreak using NAADSM. In: Canadian Association of Veterinary Epidemiology Preventive Medicine (CAVEPM) Meeting, May 29 - 30 2010, Guelph, Ontario, Canada; 2010.

[6] Portacci K, Reeves A, Corso B, Salman M. Evaluation of vaccination strategies for an outbreak of pseudorabies virus in US commercial swine using the NAADSM. In: ISVEE 12: Proceedings of the 12th Symposium of the International Society for Veterinary Epidemiology and Economics, Durban, South Africa:78; 2009.

[7] Diez Roux Ana V.. Invited Commentary: The Virtual EpidemiologistâĂŤPromise and Peril. *American Journal of Epidemiology*. 2015;181(2):100-102.

[8] HernÃąn Miguel A.. Invited Commentary: Agent-Based Models for Causal InferenceâĂŤReweighting Data and Theory in Epidemiology. *American Journal of Epidemiology*. 2015;181(2):103-105.

[9] Budgaga Walid, Malensek Matthew, Pallickara Sangmi Lee, Harvey Neil, Breidt F. Jay, Pallickara Shrideep. Predictive Analytics Using Statistical, Learning, and Ensemble Methods to Support Real-time Exploration of Discrete Event Simulations. *Future Gener. Comput. Syst.*. 2016;56(C):360–374.

[10] Zaharia Matei, Chowdhury Mosharaf, Franklin Michael J., Shenker Scott, Stoica Ion. Spark: Cluster Computing with Working Sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing:10–10; 2010.

[11] Shvachko Konstantin, Kuang Hairong, Radia Sanjay, Chansler Robert. The Hadoop Distributed File System. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST):1–10IEEE Computer Society; 2010; Washington, DC, USA.

[12] Gonzalez Joseph E., Xin Reynold S., Dave Ankur, Crankshaw Daniel, Franklin Michael J., Stoica Ion. GraphX: Graph Processing in a Distributed Dataflow Framework. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14):599–613USENIX Association; 2014; Broomfield, CO.

[13] Page Lawrence, Brin Sergey, Motwani Rajeev, Winograd Terry. *The PageRank Citation Ranking: Bringing Order to the Web.*. Technical Report 1999-66: Stanford InfoLab; 1999. Previous number = SIDL-WP-1999-0120.

[14] Wikipedia . Pareto principle — Wikipedia, The Free Encyclopedia. 2016;. [Online; accessed 25-July-2016].

[15] Shah Naman, Shah Harshil, Malensek Matthew, Pallickara Sangmi Lee, Pallickara Shrideep. Network analysis for identifying and characterizing disease outbreak influence from voluminous epidemiology data. In: :1222–1231; 2016.

[16] Sui Zhiquan, Malensek Matthew, Harvey Neil, Pallickara Shrideep. Autonomous Orchestration of Distributed Discrete Event Simulations in the Presence of Resource Uncertainty. *ACM Trans. Auton. Adapt. Syst.*. 2015;10(3):18:1–18:20.

[17] Malensek Matthew, Budgaga Walid, Pallickara Sangmi Lee, Harvey Neil, Breidt F. Jay, Pallickara Shrideep. 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:49–58IEEE Computer Society; 2014; Washington, DC, USA.

[18] Funk Sebastian, Salathé Marcel, Jansen Vincent AA. Modelling the influence of human behaviour on the spread of infectious diseases: a review. *Journal of the Royal Society Interface*. 2010;7(50):1247–1256.

[19] Paine Sarah-Jane, Gander Philippa H, Travier Noemie. The epidemiology of morningness/eveningness: influence of age, gender, ethnicity, and socioeconomic factors in adults (30-49 years). *Journal of biological rhythms*. 2006;21(1):68–76.

[20] Xiang Biao, Liu Qi, Chen Enhong, Xiong Hui, Zheng Yi, Yang Yu. PageRank with Priors: An Influence Propagation Perspective.. In: IJCAI.

[21] Aggarwal Charu C, Khan Arijit, Yan Xifeng. On Flow Authority Discovery in Social Networks.. In: SDM:522–533SIAM; 2011.

[22] Kempe David, Kleinberg Jon, Tardos Éva. Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining:137–146ACM; 2003.

[23] Hajian Behnam, White Tony. 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:497–500IEEE; 2011.

[24] Cha Meeyoung, Haddadi Hamed, Benevenuto Fabricio, Gummadi P Krishna. Measuring User Influence in Twitter: The Million Follower Fallacy.. *ICWSM*. 2010;10(10-17):30.

[25] Goldenberg Jacob, Libai Barak, Muller Eitan. 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*. 2001;2001:1.

[26] Goldenberg Jacob, Libai Barak, Muller Eitan. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*. 2001;12(3):211–223.

[27] Khrabrov Alexy, Cybenko George. Discovering influence in communication networks using dynamic graph analysis. In: Social Computing (SocialCom), 2010 IEEE Second International Conference on:288–294IEEE; 2010.

[28] Lloyd-Smith James O, Schreiber Sebastian J, Kopp P Ekkehard, Getz Wayne M. Superspreading and the effect of individual variation on disease emergence. *Nature*. 2005;438(7066):355–359.

[29] James Alex, Pitchford Jonathan W, Plank Michael J. An event-based model of superspreading in epidemics. *Proceedings of the Royal Society of London B: Biological Sciences*. 2007;274(1610):741–747.

[30] Fujie Ryo, Odagaki Takashi. Effects of superspreaders in spread of epidemic. *Physica A: Statistical Mechanics and its Applications*. 2007;374(2):843–852.

[31] Lakshman Avinash, Malik Prashant. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.*. 2010;44(2):35–40.

[32] The Apache Software Foundation . Apache HBase: A distributed database for large datasets. *http://hbase.apache.org.* ;.

[33] MongoDB Developers . MongoDB. *http://www.mongodb.org/.* ;.

[34] White Tom. *Hadoop: The definitive guide.* " O'Reilly Media, Inc."; 2012.

[35] Lam Chuck. *Hadoop in Action*. Greenwich, CT, USA: Manning Publications Co.; 1st ed.2010.

[36] The Apache Software Foundation . Hadoop. *http://hadoop.apache.org*. 2016;.

[37] Bu Yingyi, Howe Bill, Balazinska Magdalena, Ernst Michael D.. HaLoop: Efficient Iterative Data Processing on Large Clusters. *Proc. VLDB Endow.*. 2010;3(1-2):285–296.

[38] Zaharia Matei, Chowdhury Mosharaf, Das Tathagata, et al. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation:2–2USENIX Association; 2012; Berkeley, CA, USA.

[39] Jouili S., Vansteenberghe V.. An Empirical Comparison of Graph Databases. In: 2013 International Conference on Social Computing:708-715; 2013.

[40] Angles Renzo. A Comparison of Current Graph Database Models. In: Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops:171–177IEEE Computer Society; 2012; Washington, DC, USA.

[41] Ginsberg Jeremy, Mohebbi Matthew H, Patel Rajan S, Brammer Lynnette, Smolinski Mark S, Brilliant Larry. Detecting influenza epidemics using search engine query data. *Nature*. 2009;457(7232):1012–1014.

[42] Lazer David, Kennedy Ryan, King Gary, Vespignani Alessandro. The Parable of Google Flu: Traps in Big Data Analysis. *Science*. 2014;343(6176):1203–1205.

[43] Malensek Matthew, Pallickara Sangmi Lee, Pallickara Shrideep. Autonomous Cloud Federation for High-Throughput Queries over Voluminous Datasets. *IEEE Cloud Computing*. 2016;3(3):40–49.

[44] Malensek M., Pallickara S. L., Pallickara S.. Analytic Queries over Geospatial Time-Series Data Using Distributed Hash Tables. *IEEE Transactions on Knowledge and Data Engineering*. 2016;28(6):1408-1422.

[45] Tolooee Cameron, Malensek Matthew, Pallickara Sangmi Lee. A scalable framework for continuous query evaluations over multidimensional, scientific datasets. *Concurrency and Computation: Practice and Experience*. 2016;28(8):2546–2563. cpe.3651.

[46] Malensek Matthew, Pallickara Sangmi, Pallickara Shrideep. Autonomously Improving Query Evaluations over Multidimensional Data in Distributed Hash Tables. In: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC):15:1–15:10; 2013.

[47] Malensek Matthew, Pallickara Sangmi Lee, Pallickara Shrideep. Evaluating Geospatial Geometry and Proximity Queries Using Distributed Hash Tables. *IEEE Computing in Science Engineering (CiSE)*. 2014;16(4):53-61.

[48] Gao Yong, Kinoshita June, Wu Elizabeth, et al. SWAN: A distributed knowledge infrastructure for Alzheimer disease research. *Web Semantics: Science, Services and Agents on the World Wide Web*. 2006;4(3):222–228.

[49] Gonzalez Joseph E., Low Yucheng, Gu Haijie, Bickson Danny, Guestrin Carlos. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In: Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12):17–30USENIX; 2012; Hollywood, CA.

[50] Sajjad H. P., Payberah A. H., Rahimian F., Vlassov V., Haridi S.. Boosting Vertex-Cut Partitioning for Streaming Graphs. In: 2016 IEEE International Congress on Big Data (BigData Congress):1-8; 2016.

[51] Halberstam H, Laxton RR. Perfect difference sets. *Glasgow Mathematical Journal*. 1964;6(4):177–184.

[52] Petroni Fabio, Querzoni Leonardo, Daudjee Khuzaima, Kamali Shahin, Iacoboni Giorgio. HDRF: Stream-Based Partitioning for Power-Law Graphs. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management:243–252ACM; 2015; New York, NY, USA.

[53] Xie Cong, Yan Ling, Li Wu-Jun, Zhang Zhihua. Distributed Power-law Graph Computing: Theoretical and Empirical Analysis. In: Ghahramani Z., Welling M., Cortes C., Lawrence N. D., Weinberger K. Q., eds. *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc. 2014 (pp. 1673–1681).

[54] Chen Rong, Shi Jiaxin, Chen Yanzhe, Chen Haibo. PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. In: Proceedings of the Tenth European Conference on Computer Systems:1:1–1:15ACM; 2015; New York, NY, USA.

[55] Galvani Alison P, May Robert M. Epidemiology: dimensions of superspreading. *Nature*. 2005;438(7066):293–295.

[56] Shen Zhuang, Ning Fang, Zhou Weigong, et al. Superspreading SARS events, Beijing, 2003. *Emerging infectious diseases*. 2004;10(2):256–260.

[57] Woolhouse MEJ, Shaw DJ, Matthews L, Liu W-C, Mellor DJ, Thomas MR. Epidemiological implications of the contact network structure for cattle farms and the 20–80 rule. *Biology Letters*. 2005;1(3):350–352.

[58] Cortes Corinna, Vapnik Vladimir. Support-vector networks. *Machine Learning*. 1995;20(3):273–297.

[59] Ho Tin Kam. Random decision forests. In: :278-282; 1995.

[60] Fukunaga Keinosuke. *Introduction to Statistical Pattern Recognition (2Nd Ed.)*. San Diego, CA, USA: Academic Press Professional, Inc.; 1990.

[61] Kiefer J., Wolfowitz J.. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*. 1952;23(3):462–466.

[62] Pedregosa F., Varoquaux G., Gramfort A., others . Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011;12:2825–2830.

[63] Sui Zhiquan, Harvey Neil, Pallickara Shrideep. On the distributed orchestration of stochastic discrete event simulations. *Concurrency and Computation: Practice and Experience*. 2014;26(11):1889–1907.

[64] Ching Avery, Edunov Sergey, Kabiljo Maja, Logothetis Dionysios, Muthukrishnan Sambavi. One Trillion Edges: Graph Processing at Facebook-scale. *Proc. VLDB Endow.*. 2015;8(12):1804–1815.

[65] Malewicz Grzegorz, Austern Matthew H., Bik Aart J.C, et al. Pregel: A System for Large-scale Graph Processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data:135–146ACM; 2010; New York, NY, USA.

[66] The Apache Software Foundation . Apache Giraph. *http://giraph.apache.org*. ;.

[67] Low Yucheng, Gonzalez Joseph E., Kyrola Aapo, Bickson Danny, Guestrin Carlos, Hellerstein Joseph M.. GraphLab: A New Framework For Parallel Machine Learning. *CoRR*. 2014;abs/1408.2041.

[68] McCune Robert Ryan, Weninger Tim, Madey Greg. Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. *ACM Comput. Surv.*. 2015;48(2):25:1–25:39.

[69] Andreev Konstantin, Räcke Harald. Balanced Graph Partitioning. In: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures:120–124ACM; 2004; New York, NY, USA.

[70] Jain Nilesh, Liao Guangdeng, Willke Theodore L.. GraphBuilder: Scalable Graph ETL Framework. In: GRADES '13:4:1–4:6ACM; 2013; New York, NY, USA.

[71] The Apache Software Foundation . Apache Spark GraphX Source Code. *https://github.com/apache/spark/blob/master/graphx/src/main/scala/org/apache/spark/graphx/PartitionStrategy.scala*. ;.

AUTHOR BIOGRAPHIES

Naman Shah is a Master's student at Colorado State University. His research interests include network analysis, large-scale graph processing, and big data systems.
Email: namanrs@cs.colostate.edu

Matthew Malensek is an Assistant Professor in the Department of Computer Science at the University of San Francisco. Matthew received his Masters and Ph.D. degrees from Colorado State University, and his research interests are in the areas of large-scale data management and distributed systems.
Email: mmalensek@usfca.edu

Harshil Shah is a Master's student at Colorado State University. His research interests are in the areas of distributed systems, big data, and machine learning.
Email: hkshah@cs.colostate.edu

Shrideep Pallickara is a Professor in the Department of Computer Science and a Monfort Professor at Colorado State University. His research interests are in the area of large-scale distributed systems. He received his Masters and Ph.D. degrees from Syracuse University. He is a recipient of an NSF CAREER award.
Email: shrideep@cs.colostate.edu

Sangmi Lee Pallickara is an Associate Professor in the Department of Computer Science at Colorado State University. She received her Masters and Ph.D. degrees in Computer Science from Syracuse University and Florida State University, respectively. Her research interests are in the area of large-scale scientific data management. She is a recipient of the NSF CAREER award.
Email: sangmi@cs.colostate.edu