ORIGINAL RESEARCH ARTICLE UHD JOURNAL OF SCIENCE AND TECHNOLOGY

A Comparative Analysis of Spark GraphX and GraphFrames for Healthcare Data Analytics

Soz Raouf Hama[1], Alaa Khalil Jumaa[2]

[1]Information Technology Department, Technical College of Informatics, Sulaimani Polytechnic University, Sulaymaniyah, Iraq, [2]Database Technology Department, Technical College of Informatics, Sulaimani Polytechnic University, Sulaymaniyah, Iraq

ABSTRACT

Healthcare data analytics is essential for identifying disease patterns, understanding patient relationships, and supporting data-driven decision-making in modern healthcare systems. As healthcare datasets continue to grow in size and complexity, scalable graph-processing frameworks have become increasingly important for analyzing interconnected patient data. This paper compares two Apache Spark graph-processing frameworks, GraphX and GraphFrames, using the Centers for Disease Control and Prevention Diabetes Health Indicators dataset. A patient similarity graph was constructed by representing 70,692 patients as vertices and connecting highly similar patients through cosine–similarity relationships, resulting in 16,009,101 graph edges. The two frameworks were evaluated using the same graph structure and execution environment with respect to graph construction time, execution performance, memory consumption, application programming interface usability, and scalability. In addition to the framework comparison, graph-based features, including in-degree, out-degree, total degree, and PageRank, were extracted to examine the structure of the patient similarity network. The experimental results showed that GraphX completed graph construction, PageRank computation, and degree calculations considerably faster than GraphFrames. However, GraphFrames offered a higher-level programming interface, simpler integration with Spark SQL, and easier implementation of graph analytics workflows. The extracted graph measures also highlighted highly connected and structurally important patients within the network, demonstrating the usefulness of graph-based feature extraction for healthcare data analysis. GraphX completed graph construction approximately 33 times faster (92.5 s vs. 3037 s), PageRank computation approximately 780 times faster (9.1 s vs. 7132 s), and degree computation over 10,000 times faster (1.0 s vs. 10202 s) than GraphFrames, whereas GraphFrames used substantially less memory; these results indicate that GraphX is more suitable for performance-oriented graph workloads, whereas GraphFrames is advantageous when development flexibility and DataFrame integration are primary considerations.

Index Terms: In-Degree, Out-Degree, Total-Degree, Graph Analytics, GraphFrames, GraphX, Healthcare Data Analytics, PageRank, PySpark

1. INTRODUCTION

Healthcare datasets often contain large numbers of patient records with complex relationships that are difficult to analyze using traditional tabular approaches. Graph-based representations provide an effective way to model these relationships by representing patients as vertices and their similarities as graph connections. Processing such large healthcare graphs requires scalable distributed frameworks capable of handling both graph construction and graph analytics efficiently [1], [2].

To address these challenges, distributed computing frameworks such as Apache Spark have emerged as powerful solutions for large-scale data processing and analytics. Apache Spark is a distributed cluster-computing framework designed for

Access this article online

DOI: 10.21928/uhdjst.v10n2y2026.pp68-79 **E-ISSN:** 2521-4217 **P-ISSN:** 2521-4209

Corresponding author's e-mail: Soz Raouf Hama, Information Technology Department, Technical College of Informatics, Sulaimani Polytechnic University, Sulaymaniyah, Iraq. E-mail: soz.r.hama@spu.edu.iq

Received: 15-06-2026 Accepted: 19-07-2026 Published: 19-08-2026

large-scale data processing, primarily relying on in-memory computation to improve performance. It provides two key data abstractions: Resilient Distributed Datasets (RDDs) and DataFrames. RDDs represent distributed collections of objects that can be processed in parallel across a cluster using programming interfaces such as Scala, Java, Python, and R. DataFrames, on the other hand, introduce a higher-level abstraction with schema support and optimized execution, enabling more efficient handling of large datasets [3].

For graph processing, Spark provides GraphX, a graph-parallel computation framework built on top of RDDs, which represents graphs as collections of vertices and edges forming a property graph structure [4]. In GraphX, vertices and edges can store user-defined attributes, allowing the graph-based representation of real-world entities such as patients and their relationships [5]. More recently, GraphFrames was introduced as a DataFrame-based graph processing library that integrates graph analytics with Spark SQL, offering improved usability, optimized execution, and better support for large-scale structured data processing [6], [7].

The trade-offs in the design of these systems are not yet fully understood, particularly regarding their behavior on real-world graph datasets [8], [9]. In addition, it remains an open question how Spark-based graph processing frameworks compare with specialized graph-processing systems such as GraphLab [10] and PowerGraph [11].

In this study, a healthcare similarity graph is constructed from the Centers for Disease Control and Prevention (CDC) Diabetes Health Indicators dataset, where patients are represented as vertices and similarity relationships are represented as edges. The study compares GraphX and GraphFrames with respect to graph construction time, execution performance, memory consumption, application programming interface (API) usability, developer effort, and scalability. In addition, graph-based features, including degree measures and PageRank, are extracted to examine structural relationships within the patient network. The results provide practical insights into the strengths and limitations of both frameworks and support the selection of appropriate Spark-based graph-processing solutions for healthcare analytics.

The rest of this paper is organized as follows. The next section introduces the preliminaries, including Apache Spark and its graph frameworks, GraphX and GraphFrames. This is followed by the related work section. The methodology section then describes the dataset, graph construction, and graph analytics metrics such as degree measures and PageRank. Finally, the results, discussion, and conclusion sections present the experimental findings and summarize the main contributions of the study.

2. PRELIMINARIES

2.1. Graph Data Analytics

Graph data analysis is based on graph theory, where data are represented as a graph consisting of vertices and edges. Formally, a graph is defined as:

$$G = (V,E) \tag{1}$$

where (V) denotes the set of vertices (or nodes) and (E) denotes the set of edges connecting pairs of vertices. The vertices are represented as $V = \{v_1, v_2,\ldots, v_n$ while the edge set satisfies $E \sqsubseteq V \times V$, indicating that each edge represents a relationship between two vertices. Edges might have direction, which is when the relation has an orientation (for example, A refers to B), and undirected, which is when the relation is symmetric (for example, A is similar to B). Edges may carry weights that represent the strength of relationships, whereas vertices can store attributes describing the entities they represent. This structure enables a graph to capture both data and the relationships among data elements [1]. Graph analytics focuses on examining these relationships to identify important patterns within a network. Many real-world datasets, including healthcare data, contain interconnected entities where the relationships between records are as important as the records themselves. Questions such as identifying influential entities, detecting closely connected groups, and understanding how information or risk spreads across a network are naturally addressed using graph-based approaches. Although these relationships can be represented in tabular form, analyzing them often requires multiple join operations, which can become computationally expensive as the dataset grows [6], [12].

In the healthcare field, this viewpoint is especially useful. The nodes can be seen as patients, diseases, symptoms, medications, risk factors, etc., and the edges present clinically meaningful information, such as similarity between two patients, co-occurrence of two disease symptoms, patient and disease symptom, patient and risk factor relation, etc. By analyzing a cohort graph, one can identify cohorts of similar patients, surface influential or high-risk individuals through ranking algorithms, and study how comorbidities cluster, which can all support clinical decision-making and population-level analysis [13]. The structure of a simple graph, with vertices and weighted edges, is illustrated in Fig. 1 below. This generic structure directly corresponds to the patient similarity

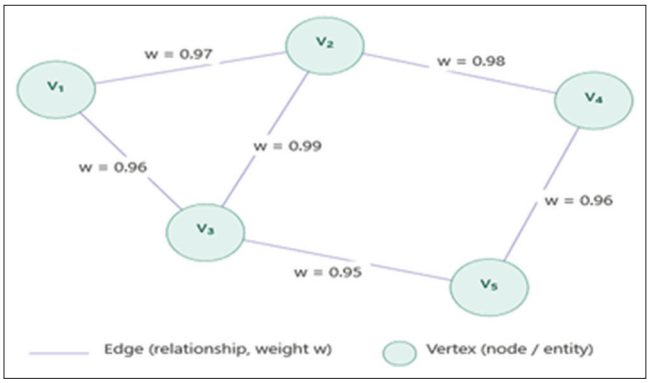

Fig. 1. A simple graph structure consisting of vertices (nodes) and weighted edges.

Fig. 2. Apache Spark architecture.

graph constructed in the methodology section, where each vertex represents a patient, and each weighted edge represents the cosine–similarity relationship between two patients.

2.2. Apache Spark

Apache Spark is an open-source distributed computing framework designed for large-scale data processing and analytics. It provides a fast and scalable platform for handling big data through in-memory computation and parallel processing. The Spark ecosystem consists of several integrated modules that support different analytical tasks. The main modules include GraphX for graph and network analysis, MLlib for machine learning algorithms, Spark SQL for structured data processing and querying, and Spark Streaming for real-time data processing [14]. Fig. 2 illustrates the major components of the Apache Spark ecosystem.

Apache Spark consists of key components, including the driver application, cluster manager, worker nodes (executors), and distributed storage systems such as Hadoop Distributed File System (HDFS). During execution, the Spark application creates a SparkContext object, which serves as the main entry point and coordinates task scheduling, resource allocation, and overall job execution across the cluster. Spark Core is the primary execution engine for Apache Spark, comprised of several subcomponents, each dedicated to specific tasks. Critical tasks, including memory management, task scheduling, and input/output operations, are handled by Spark Core. At the core of Spark's data processing model is the RDD, which represents a fault-tolerant, distributed collection of data partitioned across multiple machines. RDDs are the fundamental abstraction in Spark, supporting parallel processing in a distributed environment. They are immutable once created but can be transformed through processes that create new RDDs, enabling efficient and reliable data processing workflows [15].

2.3. Graph Analytics in Apache Spark

Many real-world datasets contain relationships between entities, such as social networks, web pages and hyperlinks, financial transactions, supply chains, telecommunication records, recommendation systems, and biological networks. Although these data can be stored in tabular form, analyzing relationships such as connectivity, neighborhood structure, and influence often requires multiple join operations, which become increasingly expensive as data volume grows. Graph processing provides a more natural representation by modeling entities as vertices and their relationships as edges. Within Apache Spark, graph-processing frameworks enable these analyses to be performed in a distributed environment, supporting scalable analysis of large connected datasets. Integrating graph processing into Spark, rather than relying on a standalone graph database, is motivated by the fact that many analytical workflows combine both graph and non-graph operations. This unified approach reduces data movement, simplifies system architecture, and allows users to work within a single processing framework and cluster environment [5], [16].

2.3.1. GraphX

GraphX graph library in Spark is the original library and was introduced in 2014 [5]. It represents a graph as a pair of RDDs, one for the vertices and one for the edges. It exposes a property-graph model whereby every vertex and edge can carry arbitrary user-defined attributes. The Pregel API [17] is at the heart of this abstraction, a vertex-centered programming model whereby computation branches off into super steps where each vertex receives messages from its neighbors, updates its own state and sends new messages on.

Numerous classical graph algorithms can be easily adapted to this model, and GraphX comes with implementations of PageRank, connected components, triangle counting, label propagation, shortest paths, and strongly connected components. GraphX also benefits from Spark's fault-tolerance and in-memory caching, important because graph algorithms are typically iterative in nature and re-traverse the same data many times. One of its limitations is that it works at the RDD level; thus, it is only available in Scala (with limited Java support), does not benefit from the improvements on the Catalyst optimizer and Tungsten execution engine that make Spark SQL so powerful, and cannot be queried with the Data Frame or SQL APIs that the rest of the Spark ecosystem [4], [18].

2.3.2. Graph frame

GraphFrames was created to address the limitations in GraphX. It uses DataFrames to represent the vertex set and the edge set instead of RDDs. This provides several immediate benefits. The API, available in Python, Scala, and R, supports queries expressed using Spark SQL. The Catalyst optimizer plans graph queries in the same way it optimizes relational queries [4]. Graph results can also be seamlessly integrated with MLlib for downstream machine learning tasks. GraphFrames introduces a capability that is not natively available in GraphX: motif finding, which provides a declarative pattern-matching mechanism for graph analysis. For example, a motif pattern such as *(a)-[e]->(b); (b)-[e2]->(c)* can be used to identify paths of length two within a graph. These patterns can be further combined with Spark SQL filters to express more sophisticated structural queries. In addition, GraphFrames provides DataFrame-based implementations of several classical graph algorithms, including PageRank, Connected Components, Triangle Counting, Breadth-First Search (BFS), Shortest Paths, and Label Propagation. This integration enables graph analytics to be performed within the Spark DataFrame ecosystem while benefiting from Spark SQL optimization capabilities [7].

In some cases, GraphFrames internally delegates operations to GraphX when the underlying algorithm benefits from a vertex-centric model. However, a key trade-off is that GraphFrames is an external package rather than part of Spark Core, meaning it must be explicitly added to the deployment. In addition, for very large iterative workloads, GraphX may still perform faster due to lower overhead compared to DataFrame-based operations [7], [18].

2.4. Cosine Similarity

Cosine similarity is an inner product measure of similarity between two non-zero vectors of an inner-product space, which measures the cosine of the angle between them. The term is based on the vector space model introduced by Salton *et al.*, in which entities are represented as vectors in a multidimensional feature space and similarity is assessed on the closeness of the orientations of the vectors, as opposed to their magnitudes. Cosine similarity is not affected by the size of instances. Since this similarity measure computes the distance based on the angle between vectors, it examines the relative distributions of feature values across two different vectors. As a result, it is widely used in information retrieval, text mining, clustering and more [19].

Given two vectors $A = (a_1, a_2,\ldots, a_n)$ and $B = (b_1, b_2,\ldots, b_n)$ in an n-dimensional space, the cosine similarity is defined as the dot product of the two vectors divided by the product of their L2 norms [19]:

$$sim(A,B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2} \times \sqrt{\sum_j b_j^2}} \quad (2)$$

where $A\ B$ denotes the dot product for A and B, and $\|A\| \cdot \|B\|$ denotes the Euclidean (L2) norm.

In the general case, the measure ranges over (−1, 1), where a value of 1 indicates that the two vectors point in exactly the same direction (maximum similarity), 0 indicates orthogonality (no similarity), and −1 indicates opposite directions. When all feature values are non-negative, as is the case for the patient feature vectors used in this work, the dot product cannot be negative, and the measure is therefore confined to the interval (0, 1), with values closer to 1 denoting greater similarity between the two entities [19].

3. RELATED WORK

This section reviews existing studies related to graph-based data analytics and distributed graph processing frameworks. It highlights relevant work on Spark-based graph processing and other large-scale graph analytics approaches to position the current study within the existing literature.

GraphX was proposed by Xin *et al.* as a graph-processing system on Apache Spark [4]. The main aim of the paper was to integrate graph-parallel computation with data-parallel execution within a single framework. The authors introduced the Resilient Distributed Graph abstraction using Spark RDDs to support graph loading, transformation,

and computation. They also implemented a vertex-cut partitioning strategy to reduce data movement across the cluster. The results indicate that, for PageRank, GraphX is faster than Mahout/Hadoop but slower than PowerGraph. Nevertheless, GraphX provides greater flexibility, fault tolerance, and simpler graph data analysis within the Spark ecosystem.

Green Fluorescent Protein (GFP-X), a parallel graph fingerprinting method for comparing massive unlabeled graphs, was proposed by Bonner *et al.* [20]. The study extracts various vertex-level and global graph features using Apache Spark and GraphX, such as PageRank, triangle count, connected components, degree, and neighborhood-based features. These features are used to formulate compact graph fingerprints and compare graphs using the Canberra distance. GFP-X was shown to scale nearly sublinearly with graph size and to handle graphs of up to 100 million vertices. According to the study, the method is sensitive to minor changes in graph topology and graph size without requiring labeled data.

The study in [9] provides a comparative review of graph-processing frameworks for large-scale graph algorithms. The research uses SNAP graph datasets to implement PageRank, Connected Components, and Triangle Counting on GraphChi, GraphX, GraphFrames, and Spark. The findings reveal that GraphChi performs better on smaller graphs, whereas GraphX becomes more robust as the graph size increases to around 100 million edges. The study also shows that GraphX incurs lower communication overhead due to its vertex-cut partitioning strategy, whereas GraphFrames is generally more expensive in terms of memory usage and network communication.

Mohammed and Jumaa proposed a methodology for the extraction, conversion, and storage of large-scale Semantic Web data using Hadoop, Apache Spark GraphX, and GFP [21]. Their approach uses SPARQL to extract data from DBpedia, where RDF triples are transformed into GraphX vertex and edge tables, and the resulting data is stored in HDFS. The results indicate that converting RDF data into GraphX format reduced the file size from 133.8 to 75.3, making it more storage efficient. The authors conclude that GraphX, as the graph-processing module of Apache Spark, together with HDFS, can effectively support the storage and management of large-scale Semantic Web data.

Apostol *et al.* proposed a new community detection algorithm framework, including Louvain, Fast Greedy, and K-Cliques, built on Spark GraphFrames [22]. The study shows that GraphFrames can achieve linear scalability and high performance for large-scale social network graphs. This is possible by combining the usability and performance of the DataFrame API with the in-memory processing capabilities of Apache Spark. These findings show that modularity-based community detection tasks can significantly benefit from using GraphFrames.

Vyakaranam *et al.* introduced GraphX as an extension of Spark that enables graph computation to be expressed naturally [23]. This work fulfills two goals: Unifying graph and data, and scaling graph computation while preserving Spark's fault tolerance, ease of use, and performance-oriented user-defined operations. The authors discuss a vertex-cut partitioning method that seeks to minimize communication in natural graphs with skewed degree distributions. GraphX offers competitive performance compared with other graph systems while also benefiting from Spark's fault tolerance, low communication overhead, and ease of use.

Parvathy *et al.* discussed the use of Apache Spark GraphX to implement BFS on big data [24]. The implementations are highly scalable and fault-tolerant for large graph traversals. GraphX is shown to perform better than GQS due to its bidirectional design. For larger datasets, the time taken for level computation in Iterative BFS is lower than that of Recursive BFS.

Theodorakopoulos *et al.* benchmarked Apache Spark GraphX against Apache Flink for graph-parallel workloads, evaluating response time, scalability, and computational complexity for algorithms including degree centrality, shortest-path computation, triangle counting, and weakly connected components [25]. Their results show that GraphX is particularly well-suited to batch, in-memory workloads, whereas Flink is preferable for streaming scenarios, reinforcing the relevance of GraphX for the batch-oriented healthcare similarity graph studied here.

Lian *et al.* examined the optimization of Apache Spark for healthcare big data management, addressing the processing of large-scale electronic health record and patient data using Spark's in-memory and distributed processing capabilities [26]. While their work does not target GraphX or GraphFrames specifically, it confirms the growing relevance of Spark-based analytics for healthcare data, motivating the graph-specific framework comparison undertaken in the present study.

The reviewed studies suggest that GraphX and GraphFrames

TABLE 1: Comparison of related work on Spark-based graph processing

Study	Framework (s)/Graph scale	Key finding
[4]	GraphX; Synthetic+real graphs (Spark Resilient Distributed Datasets)	GraphX integrates graph- and data-parallel computation; faster than Mahout/Hadoop, slower than PowerGraph
[20]	GraphX; Graphs up to 100M vertices	GFP-X scales near sub-linearly and produces compact graph fingerprints for comparison
[9]	GraphChi, GraphX, GraphFrames, Spark; SNAP graphs up to~100M edges	GraphX becomes more robust as the graph size grows; GraphFrames are costlier in memory/communication
[21]	GraphX+Hadoop/HDFS; DBpedia RDF triples	Converting RDF to GraphX format reduced file size and supported large-scale Semantic Web storage
[22]	GraphFrames; Large-scale social network graphs	GraphFrames achieves linear scalability and high performance for community detection
[23]	GraphX; Natural graphs with skewed degree distributions	GraphX offers competitive performance with low communication overhead and ease of use
[24]	GraphX; Large-scale graphs	GraphX-based BFS outperforms GQS and scales well for large traversals
[26]	GraphX, Apache Flink; Synthetic large-scale graphs	GraphX excels at batch in-memory workloads; Flink is better suited to streaming
[27]	Apache Spark (general); Healthcare big data	Confirms growing relevance of Spark for healthcare analytics (not GraphX/GraphFrames-specific)

serve different purposes within the Apache Spark ecosystem. GraphX is generally preferred for computationally intensive graph algorithms, whereas GraphFrames offers a more accessible programming interface and additional support for pattern-based graph queries. Although both frameworks have been evaluated in previous research, most studies focus on a limited set of performance measures or specific application domains. Comparisons that simultaneously examine execution time, memory usage, scalability, and development effort remain limited, particularly for healthcare data. To investigate these aspects, this study evaluates both frameworks under identical conditions using a healthcare similarity graph constructed from a real-world dataset. Table 1 shows a comparison between the related works and our current work regarding the frameworks used and the key findings of each study.

4. METHODOLOGY

This section presents the proposed work and describes the overall methodology used in this study. It begins by introducing the dataset used in the experiments, followed by an explanation of the similarity-based graph construction process. Graph analysis is then performed on the constructed graph, including the calculation of key metrics such as in-degree, out-degree, total degree, and PageRank.

4.1. Dataset Used

This study utilizes the CDC Diabetes Health Indicators dataset [25], which is derived from the Behavioral Risk Factor Surveillance System (BRFSS) 2015 survey. The dataset contains 70,692 patient records and 21 related attributes describing chronic conditions, lifestyle behaviors, healthcare access, and demographic characteristics. It is a balanced binary classification dataset with an equal 50–50 distribution of respondents without diabetes and respondents with either prediabetes or diabetes.

The target variable, Diabetes-binary, consists of two classes: 0 indicates no diabetes, whereas 1 represents prediabetes or diabetes. The balanced nature of the dataset makes it suitable for healthcare data analysis and graph-based modeling. In this work, each patient is represented as a vertex, while edges are established between patients based on the similarity of their health profiles using cosine similarity. This graph representation enables the application of graph analytics techniques and facilitates a comparative evaluation of GraphX and GraphFrames on a real-world healthcare dataset. Table 2 presents the characteristics of the dataset, whereas Table 3 summarizes the main categories of health indicators included in the dataset.

4.2. Similarity-Based Edge Construction

After representing each patient as a vertex, the next step is to establish relationships between patients based on the similarity of their health profiles. To achieve this, cosine similarity is employed to measure the similarity from the CDC Diabetes dataset. In this work, each patient is represented by a feature vector containing k attributes, which denotes the number of features used in the similarity computation.

The similarity between two patients, (pi) and (pj), is calculated using the cosine similarity formula:

TABLE 2: Characteristics of the CDC diabetes dataset

Attribute	Description
Dataset name	Diabetes binary 50/50 split health indicators BRFSS2015
Dataset size	6.35 MB
Number of records	70,692 patient records
Number of features	21 health-related attributes
Target variable	Diabetes-binary (0=No diabetes, 1=Prediabetes/diabetes)

TABLE 3: Categories of health indicators in the CDC diabetes dataset

Category	Features
Chronic conditions	HighBP, HighChol, Stroke, HeartDiseaseorAttack, DiffWalk
Lifestyle behaviors	Smoker, PhysActivity, Fruits, Veggies, HvyAlcoholConsump
Healthcare access	CholCheck, AnyHealthcare, NoDocbcCost
Demographics and health status	BMI, GenHlth, MentHlth, PhysHlth, Age, Education, Income, Sex

CDC: Centers for disease control and prevention

$$sim(P1,P2)=\frac{P1\cdot P2}{\|P1\|\cdot\|P2\|}=\frac{\sqrt{\sum_{i=1}^{n}p1_i\,p2_i}}{\sqrt{\sum_{i=1}^{n}p1_i^2}\times\sqrt{\sum_{i=1}^{n}p2_i^2}} \quad (3)$$

(Reviewer correction applied: the denominator of Eq. (2) previously repeated the summation over p_i^2; it has been corrected to read $\sqrt{\Sigma p_i^2} \times \sqrt{\Sigma p_j^2}$, consistent with Eq. (1).)

where P1and P2 denote the feature vectors of two patients, P1·P2 is the dot product of the two vectors, and ‖P1‖ and ‖P2‖ represent their Euclidean (L2) norms.

The computed similarity score is assigned as the edge weight (w). An edge is created between two patient vertices only when the similarity score exceeds a predefined threshold. This threshold was selected to retain only highly similar patient pairs and to reduce weak connections within the graph. Consequently, the resulting graph captures meaningful relationships among patients with comparable health characteristics.

Using cosine similarity offers two important advantages. First, it effectively measures similarity across multiple health indicators simultaneously. Second, it is less sensitive to differences in magnitude between patient records, making it suitable for identifying patients with similar health patterns. The resulting graph structure is subsequently used to evaluate and compare the performance of GraphX and GraphFrames through graph analytics operations such as degree computation and PageRank analysis.

In the current implementation, each patient is represented by a feature vector consisting of (k = 16) selected features. The cosine similarity score is used as the edge weight (w), and an edge is created between two patients only when the similarity score satisfies ($w \geq 0.99$). The cosine similarity threshold was empirically set to 0.99 to retain only highly similar patient pairs while maintaining a computationally manageable graph. Lower thresholds produced a substantial increase in the number of edges, resulting in significantly higher computational cost during graph construction and feature extraction.

After applying this threshold, the resulting graph contains approximately 16,009,101 similarity edges.

The BRFSS dataset has 21 health indicators besides the diabetes label, and we kept k = 16 of the health indicators for patient-to-patient similarity computation. The diagnostic target (Diabetes-binary) was excluded so that the outcome does not leak into the graph structure along with the healthcare-access and administrative items (CholCheck, AnyHealthcare, NoDocbcCost). These items are redundant with the general-health rating and represent data-collection artifacts rather than the underlying health of the patient. The 16 features retained for modeling are HighBP, HighChol, Smoker, Stroke, HeartDiseaseorAttack, HvyAlcoholConsump, GenHlth, BMI, PhysActivity, Age, Sex, Education, Fruits, Veggies, Income, and DiffWalk; These are the well-known clinical, behavioral, and sociodemographic risk factors for cardiometabolic disease [25]. The 16 features mentioned above are a collection of comorbidities, lifestyle, clinical and functional status, and demographic context. Using this complete bundle means that two patients will only be judged as similar if their full risk-factor profile is the same, which is exactly the idea of having "similar health conditions" that the edges are meant to bridge. The graph maps patients as nodes, and edges connect only the most similar patient pairs based on their health records.

4.3. Graph Construction and Graph Analytical Metrics

In this work, we constructed a patient similarity graph from the selected dataset containing 70,692 records. Each patient is modeled as a node, while edges are established based on similarity between the 16 patients' features. As we mentioned before, the Cosine similarity is used to quantify the relationship between two patients, and an edge is created only when the similarity score exceeds a threshold of 0.99.

The same graph construction process is implemented using both GraphX and GraphFrames under identical experimental settings, including dataset, feature representation, and similarity threshold, to ensure a fair comparison between the two frameworks.

Graph construction, PageRank, and degree-based feature extraction, including in-degree, out-degree, and total degree, were selected because they represent the core graph operations used in the proposed framework. Since many other graph metrics rely on the same underlying graph-processing APIs, these operations provide a representative evaluation of the computational performance of GraphX and GraphFrames.

The degree features for a directed graph $G = (V, E)$ are defined as [1]:

$$out\_degree\ (v) = |\{(v, u) \in E\}| \quad (4)$$

$$in\_degree\ (v) = |\{(u, v) \in E\}| \quad (5)$$

$$total\_degree\ (v) = \text{“}out\_degree\text{”}\ (v) + \text{“}in\_degree\text{”}\ (v) \quad (5)$$

The PageRank score is computed as [1]:

$$PR(v) = \frac{(1-d)}{|V|} + d \times \sum_{u:(u,v)\in E} \frac{PR(u)}{out\_degree(u)} \quad (6)$$

where d = 0.85 is the damping factor, $|V| = 70{,}696$, and the iteration is initialized at $PR\ (v) = 1/|V|$ for all v, repeated for 10 iterations.

The importance of each metric in healthcare data analytics is as follows:

- In-degree represents the number of patients that are highly similar to a given patient, indicating how often a patient is connected based on similar health profiles
- Out-degree represents the number of other patients to which a given patient is similar, reflecting how broadly a patient shares similar health characteristics with others
- Total degree measures the overall level of similarity-based connectivity of a patient within the population by combining both incoming and outgoing relationships
- PageRank provides a more advanced importance measure by considering not only the number of connections but also the quality of those connections.

From a healthcare perspective, patients with a high degree (in-degree or out-degree) can be interpreted as having more common or shared clinical characteristics within the population, potentially representing typical or high-prevalence health profiles. In contrast, patients with low degree values may represent more unique or less common health conditions. A high PageRank score indicates that a patient is strongly connected to other well-connected patients, suggesting that the patient belongs to a dense and highly interconnected cluster within the healthcare similarity network. This may correspond to groups of patients with similar risk profiles or shared comorbid conditions.

As shown in Fig. 3, the proposed framework proceeds through six stages. First, the CDC Diabetes Health Indicators dataset (70,692 records) is loaded, and the diagnostic target and healthcare-access attributes are excluded, retaining the 16 clinical, behavioral, and patient characteristics features used for similarity computation. Second, each patient is represented as a feature vector over these 16 attributes. Third, pairwise cosine similarity is computed between patient feature vectors using Eq. (3). Fourth, an edge is created between two patient vertices only when the similarity score satisfies $w \geq 0.99$, yielding the threshold similarity graph. Fifth, the resulting vertex and edge sets are used to construct the graph independently in GraphX (Scala/RDD-based)

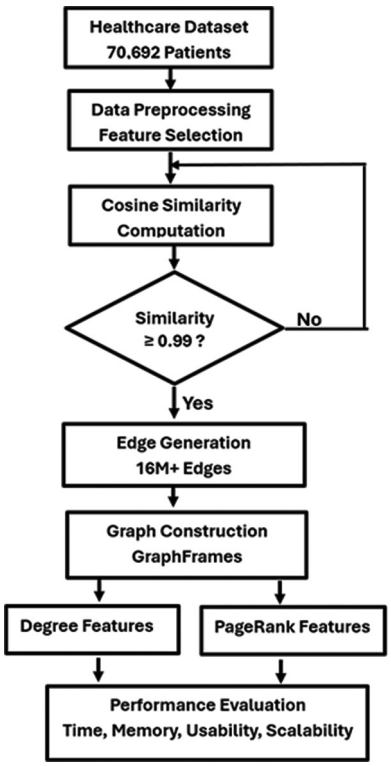

Fig. 3. Flow diagram of the proposed framework.

and GraphFrames (PySpark/DataFrame) under identical settings. Finally, degree measures (in-degree, out-degree, total degree) and PageRank are computed on the constructed graph in both frameworks, and the resulting execution time and memory consumption are compared, as reported in Section 5.

The detailed performance analysis of graph construction and graph metric computation, including execution time and memory consumption, is presented in the Results and Discussion section.

5. RESULTS AND DISCUSSION

This section presents and discusses experimental results using quantitative analysis. Both frameworks were evaluated under identical input conditions, consisting of 70,692 patient nodes and 16,009,101 edges. Therefore, any observed differences in performance can be attributed to the underlying execution models of the frameworks rather than differences in data or input configuration. A comparative analysis is conducted across five key dimensions: Execution time, memory consumption, usability, and scalability. Table 4 shows the hardware and software specifications of the experimental environment.

TABLE 4: Hardware and software specifications of the experimental environment

Category	Component	Specification
Hardware	Machine configuration	Single node (Spark local[*])
	CPU	Cori-7 2.4 GH
	Physical memory (RAM)	8 GB
	Storage	HDD, 120 GB
Software	Operating system	Linux (Debian 12)
	Apache Spark	3.5.8
	Spark install path	/opt/spark
	Deployment mode	Standalone local mode (local[*])
	GraphX implementation	Scala application programming interface (Spark Core, RDD-based)
	GraphFrames implementation	PySpark+GraphFrames package (version 4.0)
	Python	3.11.2
	Scala	2.12.15
	Java (JDK)	(OpenJDK 17)
Spark config	Driver/executor memory	8 g/8 g
	spark.driver.maxResultSize	2 g
	Shuffle partitions/ parallelism	800/800
	Serializer	Kryo (buffer. max=512 m)
	Memory fraction/storage fraction	0.7/0.3
	Adaptive query execution	Enabled (with partition coalescing)
	Local temp directory	/data/spark-temp
	RDD/DataFrame persistence	MEMORY_AND_DISK

RDD: Resilient distributed datasets

Table 5 presents a comparative evaluation of graph construction performance across GraphFrame implemented in PySpark and GraphX implemented in Scala. The comparison is conducted under identical input conditions, including the total number of nodes, the total number of edges, and a fixed similarity threshold. The main performance indicators considered in this analysis are graph creation time and memory consumption during the construction phase. This table provides a baseline understanding of the efficiency of each framework in handling large-scale graph generation from healthcare data.

Table 6 presents the execution performance comparison of key graph algorithms, specifically PageRank computation and degree analytical metrics. The performance metrics include execution time and memory usage for both PageRank and degree computation tasks.

Figs. 4 and 5 present the samples of patients identified from the healthcare similarity graph based on PageRank and Total Degree metrics, respectively. These graph-based measures were computed after constructing the patient similarity

TABLE 5: Graph construction performance comparison

Metric	PySpark (Graph Frames)	Scala (GraphX)
Total nodes	70,692	70,692
Total edges	16,09,101	16,09,101
Line code	180	290
Similarity threshold	0.99	0.99
Graph creation time	~3037 s	~92.5 s
Memory usage	~2.95 MB	~604 MB

TABLE 6. PageRank and degree-feature extraction performance

Metric	PySpark (GraphFrames)	Scala (GraphX)
PageRank time (10 L)	~7132 s	~9.1 s
PageRank memory	~15 MB	~185 MB
Degree computation time	~10202 s	~1.0 s
Degree memory	~1 MB	~70 MB

node_id	pagerank
patient_9979	311.13840698605594
patient_9981	263.55995314873064
patient_9991	247.29893274100363
patient_9999	233.75031085362275
patient_999	207.67010561747537
patient_9976	185.74987605580526
patient_9933	175.1757613857675
patient_9942	174.02275895512253
patient_9982	152.99419029050665
patient_9828	152.5898697854761

Fig. 4. Sample of patients and their corresponding PageRank scores.

node_id	total_degree
patient_31101	303
patient_34608	215
patient_35170	534
patient_306	1406
patient_33914	869
patient_34827	782
patient_35168	616
patient_15929	675
patient_18669	252
patient_21786	900

Fig. 5. Sample of patients and their corresponding total degree values.

network and extracting graph features using GraphFrames. Fig. 4 shows the patients with the related PageRank scores, whereas Fig. 5 shows the patients with the total degree values. Together, these figures provide a visual summary of the most prominent nodes in the network according to each graph metric. A detailed interpretation of these findings is presented in the Discussion section.

The experimental findings demonstrate noticeable differences between GraphX and GraphFrames across several performance aspects, including execution time, memory footprint, usability, and scalability. These results provide a comprehensive basis for evaluating the suitability of both frameworks for large-scale healthcare graph analytics.

Execution time: A significant performance gap is observed in execution time across all evaluated operations. GraphX consistently outperformed GraphFrames, achieving approximately ×33 faster graph construction (92.5s vs. 3037s), around ×780 faster PageRank computation (9.1s vs. 7132s), and over ×10,000 faster degree computation (1.0s vs. 10202s) on the same 70,692-node and 16-million edge graph. This performance advantage is primarily attributed to GraphX's vertex-centric Pregel-based execution model and its tight integration with Spark's RDD layer. In contrast, GraphFrames introduces additional overhead due to its DataFrame abstraction layer and Python–JVM serialization costs in PySpark environments. These results are summarized numerically in Table 7.

Memory footprint: The memory behavior exhibits an inverse trend compared to execution time. GraphX requires higher memory consumption due to its RDD architecture, where Vertex_RDD and Edge_RDD are materialized in memory. Specifically, it consumes approximately 604 MB during graph construction, 185 MB and 70 MB during PageRank and degree computations, respectively. In contrast, GraphFrames maintains a significantly lower memory footprint, with approximately 2.95 MB for graph construction, 15 MB for PageRank, and 1 MB for degree computation. This efficiency is achieved through Catalyst-optimized DataFrame execution, which relies more on lazy evaluation and disk spilling strategies. These memory figures are likewise summarized numerically in Table 7.

Table 7 presents a comprehensive comparison between GraphX and GraphFrames across multiple evaluation dimensions. These include graph construction time, execution time for PageRank and degree computation, memory footprint, API and language support, motif and pattern query capabilities, developer effort, and suitability for different workload types.

Usability and developer effort: GraphFrames demonstrates a clear advantage in terms of usability and development effort. Its high-level DataFrame and Spark SQL interface reduces code complexity and enables faster development cycles compared to GraphX, which relies on lower-level RDD programming. In addition, GraphFrames benefits from Catalyst query optimization, reducing the need for manual optimization strategies such as explicit caching or partition tuning. This makes GraphFrames particularly suitable for exploratory analytics and ML pipelines, especially in mixed language environments involving Python and Spark MLlib.

Scalability with dataset size: The observed results at the scale of 70,692 nodes suggest consistent trends under increasing

TABLE 7: Comprehensive comparison of GraphX and GraphFrames across performance, memory, and functional capabilities

Criterion	GraphX (Scala)	Graph frames (PySpark)
Graph construction time (sec)	92.5 (faster)	3037
PageRank execution time (sec)	9.1 (faster)	7132
Degrees execution time (sec)	1.0 (faster)	10202
Memory footprint (MB)	Higher (~604 construction)	Lower (~2.95 construction)
API/language support	Scala/limited Java, RDD-based	Python, Scala, R; data frame and Spark SQL
Motif/pattern queries	Not supported natively	Supported (declarative motif finding)
Developer effort/learning curve	Higher (low-level RDD API)	Lower (high-level data frame API)
Best-fit workload	Iterative, memory-intensive graph algorithms	Motif querying and ML-pipeline integration

API: Application programming interface, RDD: Resilient distributed datasets

dataset sizes. GraphX scales efficiently in memory-rich environments, with execution time increasing approximately linearly with edge growth, provided the graph remains fully in memory. However, performance degrades sharply once memory limits are exceeded. GraphFrames, on the other hand, provides more stable but slower scaling behavior due to its DataFrame execution model and Catalyst optimization overhead. It also benefits from disk spilling mechanisms, making it more robust for very large datasets but consistently slower in iterative workloads. Therefore, system design choices should prioritize GraphX for computing-intensive iterative analytics, while GraphFrames is more appropriate for mixed workloads involving motif querying and machine learning integration.

Graph-Based Patient Network Analysis: The results presented in Figs. 4 and 5 provide additional insight into the structure of the healthcare similarity network. Patients with the highest total degree exhibited strong similarity connections with a large number of other patients, indicating that their health profiles closely matched those of many individuals in the dataset. Similarly, patients with high PageRank scores occupied central positions within the network and were connected to other highly connected patients. The presence of highly ranked patients according to both metrics suggests the existence of dense clusters of individuals sharing common health characteristics. These findings demonstrate the value of graph-based measures for identifying representative and structurally important patients within large-scale healthcare datasets.

Overall, the results confirm the main hypothesis of this work: GraphX and GraphFrames are not interchangeable, and neither framework is universally superior. Instead, they occupy different positions along the performance usability trade-off spectrum. While GraphX provides superior performance for large-scale iterative graph analytics, GraphFrames offers greater development flexibility and integration with Spark's DataFrame ecosystem. In addition, the graph-based analysis demonstrated that measures such as total degree and PageRank can provide meaningful insights into patient similarity relationships and network structure. Therefore, the choice of framework should be guided by both the computational requirements and the analytical objectives of the application.

6. CONCLUSION

This paper presented a comparative analysis of GraphX and GraphFrames, two graph processing frameworks built on Apache Spark, using the CDC Diabetes Health Indicators dataset as a healthcare case study. A patient similarity graph consisting of 70,692 nodes and 16,009,101 cosine-similarity edges was constructed, and the experimental results show a clear performance-usability trade-off: GraphX completed graph construction, PageRank computation, and degree computation approximately 33 times, 780 times, and over 10,000 times faster than GraphFrames, respectively, whereas GraphFrames consumed substantially less memory and offered a higher-level, more usable API. Graph-based measures such as in-degree, out-degree, total degree, and PageRank extracted from the network provided valuable insights into patient relationships, identifying highly connected patients and revealing groups of patients with similar health profiles. Overall, the findings confirm that neither framework is universally superior; instead, each serves different analytical requirements depending on system constraints and application goals. GraphX is more appropriate for performance-critical and large-scale iterative computations, while GraphFrames is more suitable for flexible, SQL-integrated, and ML-oriented workflows.

In future research, we plan to extend this work by focusing on feature extraction from graph-structured healthcare data and integrating these features into machine learning models

for disease prediction and risk classification. In particular, graph-derived features such as centrality measures and similarity-based embeddings will be further explored to improve predictive accuracy in healthcare analytics systems.

REFERENCES

1. M. Newman. "*Measures and metrics*". In: Networks. 2nd ed., Ch. 7. Oxford University Press, Oxford, UK, 2018.
2. W. Raghupathi and V. Raghupathi. "Big data analytics in healthcare: Promise and potential". *Health Information Science and Systems*, vol. 2, p. 3, 2014.
3. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker and I. Stoica. *"Resilient distributed datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing*". 9th USENIX Symposium on Networked Systems Design and Implementation, San Jose, CA, 2012.
4. R. S. Xin, Joseph E. Gonzalez, M. J. Franklin and I. Stoica. "GraphX: A Resilient Distributed Graph System on Spark". Association for Computing Machinery, New York, 2013.
5. J. E. Gonzalez, R. S. Xin, A. Dave and D. Crankshaw. "*GraphX: Graph Processing in a Distributed Dataflow Framework*". USENIX Association, Berkeley 2014.
6. A. Dave, A. Jindal, E. L. Li, R. Xin, J. Gonzalez and M. Zaharia. "*GraphFrames: An integrated API for mixing graph and relational queries*". In: ACM International Conference Proceeding Series, Association for Computing Machinery, 2016.
7. R. K. Mishra, S. R. Raman. "*GraphFrames"*. In: PySpark SQL Recipes. Apress, Berkeley, CA, 2019.
8. K. Ammar and M. T. Özsu. "Experimental analysis of distributed graph systems". *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1151-1164, 2018.
9. S. Mary Arul, G. Senthil, S. Jayasudha, A. Alkhayyat, K. Azam and R. Elangovan. "Graph Theory and Algorithms for Network Analysis". *E3S Web of Conferences*, vol. 399, p. 08002, 2023.
10. Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin and J. M. Hellerstein. "*GraphLab: A New Framework for Parallel Machine Learning*". In: Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI), 2010, pp. 340-349.
11. J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin."*PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs*". In: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12), 2012, pp. 17-30.
12. M. Dayarathna and T. Suzumura. "*Benchmarking Graph Data Management and Processing Systems: A Survey*". [arXiv Preprint], 2021.
13. M. Bin Hannan Siam, M. R. Khan, M. F. Elahe, M. S. Arman and S. Akter. "Effects of similarity networks in graph-based multi-omics classification". PLoS One, vol. 21, no. 3, p. e0344754, 2026.
14. M. Zaharia, R. S. and Wendell, P. Das, T. Armbrust, M. Dave, A. Meng, X. Rosen, J. Venkataraman, S. Franklin, M. J. Ghodsi, A. Gonzalez, J. Shenker, S. S. Ion. "Apache spark: A unified engine for big data processing". *Communications of the ACM*, vol. 59, no. 11, pp. 56-65, 2016.
15. H. Khayrolla Omar, A. K. Jumaa. "Distributed big data analysis using spark parallel data processing". *Bulletin of Electrical Engineering and Informatics*, Vol 11, no 3, pp. 1505-1515, 2022.
16. S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin and M. T. Özsu. "The ubiquity of large graphs and surprising challenges of graph processing". Proceedings of the VLDB Endowment, vol. 11, no. 4, pp. 420-431, 2017.
17. G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. "*Pregel: A system for large-scale graph processing*". In: A. Elmagarmid and D. Agrawal, Eds. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. New York, USA: ACM, 2010, pp. 135–146.
18. M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi and M. Zaharia. "*Spark SQL: Relational data processing in Spark*". In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, pp. 1383-1394, 2015.
19. C. D. Manning, P. Raghavan and H. Schütze. "*An Introduction to Information Retrieval*". Cambridge University Press, Cambridge, 2009.
20. S. Bonner, J. Brennan, G. Theodoropoulos, I. Kureshi and A. S. McGough. "*GFP-X: A parallel approach to massive graph comparison using spark*". 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 2016, pp. 3298-3307.
21. W. M. S. Mohammed and A. K. Jumaa. "An efficient approach to extract and store big semantic web data using hadoop and apache spark GraphX". *Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 13, e31506, 2024.
22. E. S. Apostol, A. C. Cojocaru and C. O. Truică. "*Large-Scale Graphs Community Detection using Spark GraphFrames*". In: Proceedings 2024 23rd Proceedings of an International Symposium. Parallel and Distributed Computing (ISPDC), 2024, pp. 1-5.
23. L. Vyakaranam, R. Raman, H. Shah, N. Mishra, R. Meenakshi and S. Murugan. "*Simplifying Graph-Parallel Computation in Apache Spark with GraphX*". In: 2024 8th International Conference on Electronics, Communication and Aerospace Technology (ICECA), IEEE, 2024, pp. 763-769.
24. P. R. Parvathy, J. Lenin and S. Mishra. "Implementing apache spark GraphX in big data using breadth-first search". *Innovations in Intelligent Systems and Advanced Engineering*, 1(1), 19-28,2025.
25. L. Theodorakopoulos, A. Karras, A. Theodoropoulou and G. Kampiotis. "Benchmarking big data systems: Performance and Decision-making implications in emerging technologies". *Technologies*, vol. 12, no. 11, p. 217, 2024.
26. Z. Lian, X. Lin, and L. Yin. "Optimizing apache spark for healthcare big data management". *Transactions on Computer Science and Intelligent Systems Research*, vol. 7, pp. 113-118, 2024.
27. A. Teboul. "*Diabetes Health Indicators Dataset (BRFSS 2015)*". Kaggle, San Francisco, 2021. Available from: https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset [Last accessed on 2026 May 25].