Innovations in Intelligent Systems and Advanced Engineering
Vol. 1, No. 1, 2025 (February)

Implementing Apache Spark GraphX in Big Data Using Breadth-First Search

Palaniraj Rajidurai Parvathy
Mphasis Corporation, Chandler, Arizona, USA
palanirajrps@gmail.com

J. Lenin
Department of Computer Science & Engineering, Alliance College of Engineering and Design, Alliance University, Bangalore, Karnataka, India
lenin.j@alliance.edu.in

Suman Mishra
Department of Electronics & Communication Engineering, CMR Engineering College, Hyderabad, India
emailssuman@gmail.com

Abstract: Application of Apache's Breadth-First Search (BFS) to Big Data with order to efficiently traverse graphs with massive datasets, Spark GraphX intends to make use of Apache Spark's robust distributed processing capabilities. Focusing on scalability and speedup in processing vast graph structures, the purpose is to show that utilizing Spark GraphX for BFS is feasible and has performance advantages. By showcasing the benefits of fault tolerance and parallel processing that are intrinsic to the Spark environment, we want to provide a thorough technique for installing BFS in Spark GraphX. Applications for this method might include bioinformatics, recommendation systems, and social networks, all of which deal with large graphs. Exploring the possibility of handling complicated graph queries with substantial gains in processing speed and resource usage is possible via the integration of BFS with Spark GraphX. This advances the capabilities of big data analytics in graph processing. In one example of Node Distance from Source, the numbers vary from 32 to 93 kilometers, while in another case with 5 nodes linked, the results range from 14 to 78 kilometers. When comparing the two sets of data, the values are 23 to 98 kilometers. These ranges are all derived from the Sample Graph Data.

Keywords: Breadth-first search, Apache spark, GraphX, big data analytics, graph traversal, distributed processing

I. INTRODUCTION

To effectively analyze and handle massive volumes of data, sophisticated computing tools have become essential with the rise of big data. Apache Spark is one such technology; it is an open-source unified analytics engine designed specifically for processing large data. When it comes to graph processing and parallel computing, Spark's GraphX API is second to none. In addition to facilitating the large-scale execution of graph algorithms, GraphX also permits the representation and manipulation of graphs. Graph theory's BFS method is essential for many tasks, such as web crawling, network routing, and social network research. Apache Spark GraphX is designed to take use of Spark's distributed computing capabilities so that massive datasets may be efficiently processed using graphs. Integrating GraphX with Spark makes sophisticated calculations feasible, even on conventional single-machine systems, and it becomes feasible to manage graphs with billions of edges and vertices. In this setting, we want to use BFS to traverse graphs and discover interesting things in massive data.

Learning everything there is to know about implementing BFS with Apache Spark GraphX is the main goal. Building the graph structure, starting the BFS algorithm, and optimizing the computation for efficiency and scalability are all part of this. Assessing how well the BFS implementation in GraphX handles large-scale datasets is another important goal. Time spent computing, resources used, and precision of BFS traversal outcomes are all metrics to track here. Executing graph algorithms on large data platforms is the ultimate objective of this demonstration of Apache Spark GraphX's practical use. By zeroing in on BFS, we want to demonstrate how this method may be used to address practical issues involving exploring and traversing graphs. Our goal is to lay out a solid plan for integrating BFS into GraphX so that other graph algorithms in the Spark ecosystem may be explored and used.

The whole BFS implementation in Apache Spark GraphX, from ingesting graph data to running and evaluating the algorithm, is within the purview of this scope. That encompasses... Getting massive datasets ready for graph construction in GraphX by loading and parsing them and checking for proper formatting. Configuring the BFS algorithm in GraphX, which involves establishing the traversal's data structures and identifying the source vertex. Using Spark's distributed processing features to coordinate computation over several nodes, execute the BFS algorithm on the generated graph. Improving the algorithm's performance while decreasing its resource usage is also part of this stage. Examining the BFS traversal's output, testing the implementation's efficacy, and checking the findings against benchmark datasets to ensure correctness and performance. To show how useful GraphX is for managing large data problems, the BFS implementation to real-world datasets like web graphs or social networks and extract useful insights is applied. Section 2 covers Apache Spark GraphX with Breadth-First Search for Big Data. Apache Spark GraphX is made available by Sector 3 using Breadth-First Search. Apache Spark GraphX employs Breadth-First Search across several datasets, as shown in Section 4. Finally, in Section 5 this ends with a conclusion.

II. LITERATURE REVIEW

Subgraphs are called Connected Components if any two nodes in the graph may be accessed from any other node in the graph. The usual approach to computing linked components is to use BFS or Depth-First Search (DFS) to traverse all the edges and vertices in a graph, which usually results in a linear time complexity. In distributed systems, however, optimizing computation involving linked components mostly involves addressing issues with centralized workload and communication at high-degree vertices [1]. Graph algorithms help monotonic algorithms like Connected Components and Breadth-First Search to analyze graphs incrementally by recording lightweight relationships between results. The three types of incremental algorithms that Aion can handle are as follows: (i) non-holistic aggregations like average node values or relationship properties; (ii) monotonic path-based algorithms like Breadth-First Search (BFS) or Single-Source Shortest Path (SSSP); and (iii) non-monotonic algorithms that can converge to correct results without relying on node initialization, like PageRank or Graph Coloring [2].

The bottom-up search for repeating substructures in Apriori-based techniques starts with small size graphs. Each cycle involves combining two similar but somewhat different substructures to enlarge the current substructures. To find the subgraph lattice in each graph dataset, an Apriori-based method employs a generate-and-test approach using a BFS [3]. Instead of BFS/DFS, a best-first search technique is suggested to produce highly cohesive partitions. To adaptively tweak the partition results, sophisticated subgraph-local search techniques are used. Best-First Search is an improvement over conventional BFS and DFS that combines degree balanced and boundary generation. While exploring graphs, BFS keeps track of previously expanded frontiers in a frontier queue and adds new ones as they become available [4].

To make data mining more efficient in general, our algorithm employs the BFS method. Addressing candidate creation is the primary challenge in developing a BFS algorithm for serial episodes involving simultaneous occurrences. The BFS method is used by one group of algorithms for mining episodes that occur often [5]. We cover a variety of methods for optimizing BFS with SYCL, as well as our algorithms for graph creation and partitioning, and we provide relevant dataset information. The two distinct designs are shown in the same section. With the goal of methodically exploring all a network's nodes, BFS is a graph traversal algorithm [6]. When applied to puzzle games, the Breadth-First Search method provides a quicker solution that doesn't need much thinking. Even in terms of processing time, this algorithm optimizes game-solving performance by solving issues and providing a step-by-step solution. The BFS method has been used to solve these issues in related studies before. Using the BFS algorithm to solve puzzle games gives players an edge by providing optimum solutions in terms of number of steps to completion and capacity to handle various problem-specific situations [7]. So far, the IPU has only undergone evaluations using the BFS algorithm. Discovering a network "level by level" means visiting every node's neighbor before going on to the next level. This basic graph traversal technique is known as the BFS algorithm. It is an essential part of several different graph algorithms and is most often used to locate the shortest route in an unweighted graph. Unweighted directed (or undirected) networks may have BFS locate all its nodes [8].

Rectangle searches within and beyond the room, even re-expanding worthless states when it finds better pathways to them, whereas GBFS swiftly fills each minimum and achieves the objective. Exhausting all states that seem better than the exits is the goal of a best-first search, which is the limiting case of ARA* and AEES [9]. This search will cover the complete local minimum. In contrast, BFS takes a broad approach to route exploration. These algorithms are both steppingstones to learning more complex search approaches and excellent examples of their kind. To find the shortest route for a mobile robot to navigate through several mazes, DFS was contrasted with other algorithms including BFS and A∏ [10].

Nodes must notify boundary information to synchronize the search progress when using graph search algorithms, like BFS, in a distributed context. Example: in a distributed computing setting, the distributed BFS approach methodically explores the network and uncovers connections among its nodes [11]. Ensuring the power grid is safe, stable, and economically operated is the responsibility of the power communication network. It plays a crucial role in the electricity system by relaying data from mains-powered devices and notifying of problems. In addition to facilitating power transfer operations, the power communication network is essential for power grid diagnostics and problem management [12]. This is especially true for large-diameter graphs with BFS-like primitives: the cost of scheduling and syncing threads between them increases as the number of rounds increases while traversing the graph in BFS order. The high expense of creating and synchronizing threads between rounds is a significant barrier to employing parallel BFS or other methods, particularly for large-diameter graphs [13]. More and more individuals are opting to drive as their disposable income rises. Accidents may happen and passengers' safety isn't guaranteed due to the erratic driving tendencies of conventional vehicle drivers. Each one of them was able to effectively conduct autonomous vehicle study after a lengthy time of technological investigation and accumulation [14]. While structurally like the DFS environment, this one differs in its primary algorithmic requirement, telling the model to use the BFS method to navigate graphs. The BFS environment can evaluate the model's understanding of the first-in-first-out queue principle, a basic feature of BFS, thanks to this important difference [15]. Using the inference rules at any point within the formulas is possible using deep inference, a proof-theoretic approach. The sequent calculus's shallow inference, which allows rule instances only at the top-level formulas, is extended by deep inference. A rich proof theoretical analysis is revealed by deep inference, which offers fresh views for different logics, due to the combinatoric abundance of conceivable rule instances [16].

An effective pruning strategy for breadth-first search in the relevant Hasse diagram is suggested for performance analysis, with the goal of finding its dominating critical syndromes. Next, the weight threshold for the weighted voting rule is designed using this analysis. When compared to anonymous voting and group selection rules, the weighted voting rule with proposed threshold performs better in simulations [17]. Models created directly from image analysis of porous and fractured medium, as well as any system of linear equations with singular values, are susceptible to this problem; it is not limited to diluted regular networks. Finding and removing solitary locations and clusters has been accomplished using several iterative search techniques to tackle this singularity [18]. Beginning at any given node, the BFS will discover all additional nodes that may be reached from this node. Each of these nodes is part of a single linked part. Then, BFS begins the search process all over again, this time at a node that has not been visited before. We keep going until we've visited every single node. There are no longer any edges that connect the recognized components; all of them have been located and identified. Therefore, we compare BFS, which effectively divides-and-conquers large sparse networks without sacrificing accuracy, with full certainty that the actual clusters were not divided for the provided network [19]. To maximize the efficacy of decisions made in uncertain and dynamically changing battlefield situations while also ensuring the optimal use of military resources, it is necessary to develop requirements for the system that will be based on AI methods [20].

III. MATERIALS AND METHODS

Among the most basic graph traversal algorithms is BFS, which, before proceeding to the nodes at a deeper level, investigates all the neighbors at the current depth. This explores the many approaches used by BFS to show how it may be used in different fields. Instead of being a cookie-cutter algorithm, BFS provides versatility by way of a variety of strategies that are fine-tuned to meet the needs of individual applications. There are a variety of BFS methods, each with its own set of benefits and drawbacks; they include iterative, recursive, bidirectional, and parallel BFS. The aim is to demonstrate BFS's flexibility to various situations and issue complexity by looking at these methods. The goal is to provide a detailed introduction to all the different BFS methods, with an emphasis on their uses and consequences in the actual world. The goal of this investigation is to learn more about BFS and how it helps with addressing graph traversal difficulties in various contexts. A job is an RDD activity called by a Spark application is shown in Figure 1. The Spark scheduler constructs a DAG and starts a task for RDD actions.

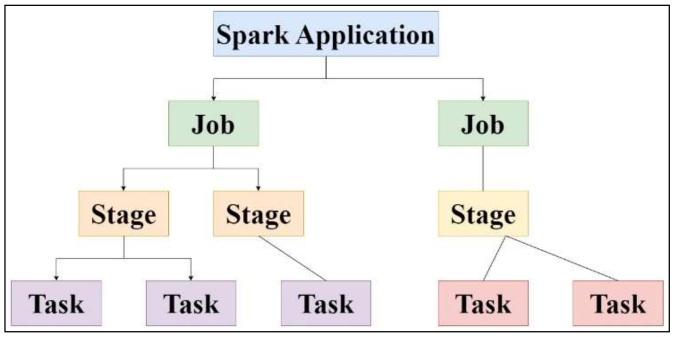

Figure 1: The Spark application tree

3.1. Iterative BFS

This method repeatedly explores nodes using a queue data structure. It is the most popular BFS implementation and suited for most situations. Iterative Breadth-First Search (BFS) is a basic network traversal technique that checks nodes level by level. It sequentially visits neighbor nodes at the current depth before going on to the next depth. This iterative technique is used in shortest route discovery, connectivity analysis, and graph traversal to efficiently explore graphs. In this study, the iterative BFS algorithm's concepts, uses, and consequences for graph theory and computer problem-solving is explained. Spark schedulers construct DAGs and start jobs when RDD actions are invoked. Each work is staged. One step is a wide transformation, which happens when a reducer function stirs RDD partitioned data throughout the network is illustrated in Figure 2. Each step has parallel jobs that execute the same executor command. The number of RDD divisions and calculation outputs determine stage tasks.

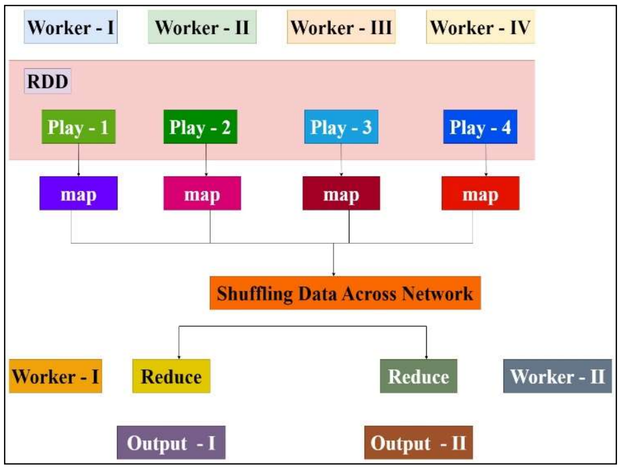

Figure 2: In a Spark cluster with four workers, each worker maintains one RDD partition and executes a map job. Two reduced tasks on two workers handle data from the map tasks

3.2. Recursive BFS

This variation implements BFS recursively instead of iteratively. Recursively exploring each graph level may cause stack overflow faults for big graphs. Using recursion, Recursive Breadth-First Search (BFS) traverses nodes level by level. Recursive BFS traverses the graph via function calls, unlike iterative BFS, which stores and manages nodes in a queue. This BFS implementation viewpoint may provide benefits in certain cases. Recursive BFS is discussed, including its concepts, uses, and benefits over iterative BFS in graph traversal and related computing issues. Table 1 details BFS in Apache Spark GraphX for Big Data. The Algorithm role efficiently begins and traverses the complicated network structure, simplifying and scaling large-scale analytics activities. The Platform helps workers share tasks, improving distributed system efficiency and speed. The framework optimizes graph representation for flexible and customized graph processing. In cloud computing settings, infrastructure optimizes resource use and costs via dynamic resource allocation. These elements work together to provide Apache Spark GraphX-enabled BFS big data analytics.

Table 1: Insights Into Breadth-First Search Implementation in Big Data Using Apache Spark Graphx

Aspect	Role	Function	Benefit	Scope
Algorithm	Breadth-First	Traversal of graph nodes in layers	Efficiently discovers shortest paths and distances	Applied to large-scale graph datasets
Framework	Apache Spark	Utilizes distributed computing for parallel processing	Handles massive datasets with scalability	Suitable for big data environments
Data Structure	GraphX	Represents data as distributed property graphs	Enables efficient storage and manipulation of graph data	Facilitates graph-based computations
Optimization Technique	Parallelism	Executes BFS in parallel across distributed nodes	Accelerates computation time by leveraging parallelism	Enhances performance in distributed systems
Scalability	Distributed	Distributes computation across multiple nodes	Scales seamlessly with growing data volumes	Adaptable to increasing data sizes and loads

3.3. Bidirectional BFS

Bidirectional BFS analyzes the graph from both start and end nodes, meeting in the center. This method is effective for determining the shortest route between two unweighted network nodes. Figure 3 shows the algorithm structure.

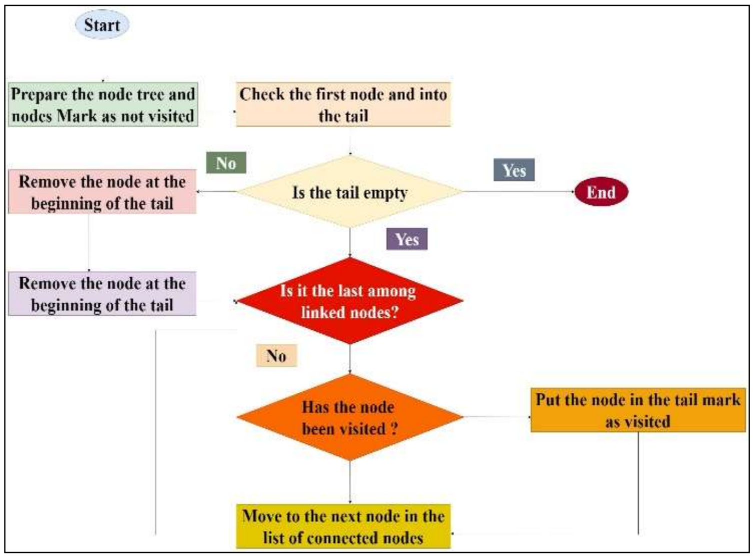
Start
Prepare the node tree and nodes Mark as not visited
Check the first node and into the tail
No
Yes
Remove the node at the beginning of the tail
Is the tail empty
End
Yes
Remove the node at the beginning of the tail
Is it the last among linked nodes?
No
Has the node been visited ?
Put the node in the tail mark as visited
Move to the next node in the list of connected nodes

Figure 3: Breadth-first search algorithm flow chart

The specialized graph traversal method Bidirectional Breadth-First Search (BFS) meets in the middle of a graph from the start and finish nodes. This method is better than BFS for finding the shortest route between two nodes in unweighted networks. Bidirectional BFS minimizes search space and processing cost by investigating the graph from both ways, speeding traversal. This discusses bidirectional BFS, its concepts, applications, and efficacy in addressing shortest route and graph traversal issues. The Breadth-First approach queues all linked nodes from the start node and initially labels them as ‘not visited’. After connecting the first node to the queue, the algorithm traverses the linked nodes to record distances. The additional nodes are then processed similarly. The end nodes are marked as ‘visited’ after each visit.

IV. RESULTS AND DISCUSSIONS

4.1. Layered BFS

Nodes are stacked by distance from the start node in layered BFS. This method visualizes graph structure and aids graph analysis. Layered Breadth-First Search (BFS) groups nodes into layers by distance from the start node. Each layer represents nodes at a given depth from the start node, enabling organized graph exploration. Layered BFS shows the graph's hierarchical structure and speeds up traversal by concentrating on nodes at each level. This discusses layered BFS, its concepts, applications, and consequences for graph analysis and computer problem-solving. Where $G=(V,E)$ is the graph with vertices V and the edges E, s is the source vertex, d is the depth level. D is the maximum depth level (the longest shortest path from s to any reachable vertex). $dist(s,v)$ is the shortest path distance from s to vertex v. Starting from the source vertex, a vertex in the visited set visits the complete layer of unvisited vertices and adds recently visited vertexes. Consider vertexes as Micromouse maze cells to compare BFS. Micromouse's BFS algorithm labels cells scanning from the initial cell to all nearby neighbours. The first cell is zero. The programme tracks start cell's immediate neighbours. BFS finds the shortest route. Continue until the objective is found. Figure 4 displays the algorithm flow chart.

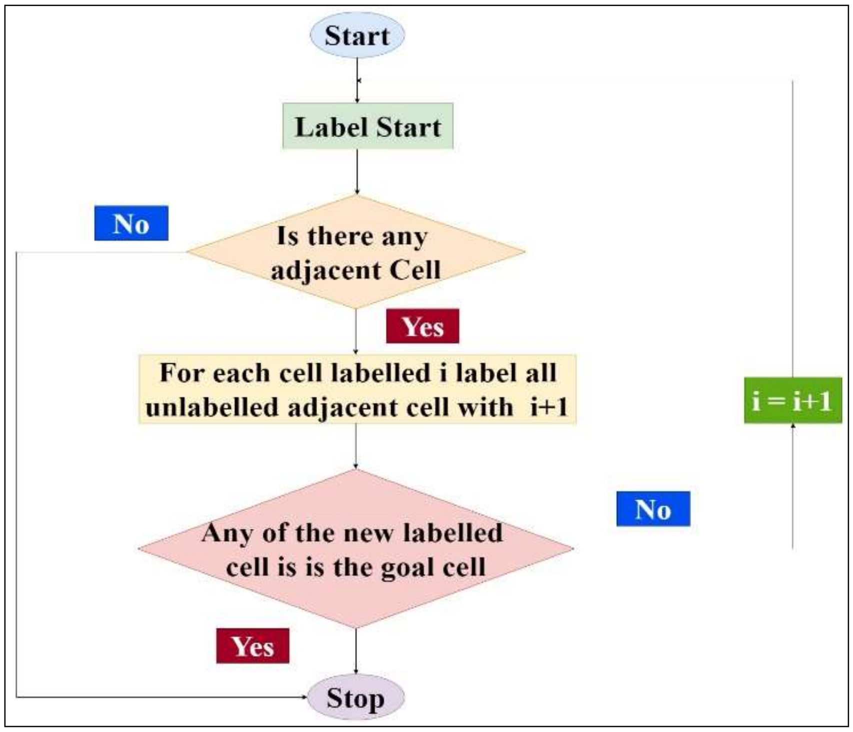
Start
Label Start
Is there any adjacent Cell
No
Yes
For each cell labelled i label all unlabelled adjacent cell with i+1
i = i+1
No
Any of the new labelled cell is is the goal cell
Yes
Stop

Figure 4: Flow Chart for BFS Algorithm

4.2. Pseudo code for Breadth First Search

```
BFS(graph, start_node):
  create a queue Q
  mark start_node as visited
  enqueue start_node into Q

  while Q is not empty:
    current_node = dequeue from Q
    process current_node

    for each neighbor in neighbors of current_node:
      if neighbor is not visited:
        mark neighbor as visited
enqueue neighbor into Q
```

Explanation

Initialize a Queue: BFS uses a queue to keep track of nodes that need to be explored. This ensures nodes are processed in the order they are discovered. Mark the Start Node: The starting node is marked as visited to prevent revisiting it. Enqueue the Start Node: The starting node is added to the queue, making it the first node to be processed. Process Nodes: While the queue is not empty, repeat the following steps:

- **Dequeue a Node:** Remove and retrieve the front node from the queue. This node is currently being processed.
- **Process the Node:** This could involve any operation, such as printing the node's value.
- **Check Neighbors:** For each adjacent (neighboring) node:
    - If the neighbor hasn't been visited, mark it as visited to avoid revisiting.
    - Enqueue the neighbor, adding it to the queue for subsequent processing.
- **Repeat:** Continue this process until the queue is empty, indicating all reachable nodes have been processed.

Table 2 compares Apache Spark GraphX BFS in Big Data. The basic graph traversal method BFS is used for shortest route finding and graph exploration.

Table 2: Implementing Breadth-First Search in Big Data Using Apache Spark GraphX

Aspect	**Uses**	**Advantages**	**Application**	**Shortcomings**
Algorithm	Traversing graphs	Scalability, parallel processing	Social network analysis, web crawling	May require optimization for specific graph structures
Apache Spark	Distributed computing platform	Fault tolerance, in-memory processing	Big data analytics, machine learning	Steeper learning curve, resource-intensive
GraphX	Graph processing	Graph parallelism, vertex-centric API	Graph-based algorithms, network analysis	Limited to graph-based operations, overhead for non-graph tasks
Breadth-First Search	Shortest path finding, graph exploration	Uniform level-wise exploration, optimal for unweighted graphs	Social network analysis, web crawling	Inefficient for large graphs with irregular structures
Scalability	Handling large datasets, distributed computing	Horizontal scaling, fault tolerance	Large-scale graph processing, distributed graph analytics	Potential performance degradation with increasing cluster size

For large data analytics, Apache Spark, a distributed computing platform, enables fault tolerance and in-memory processing. An Apache Spark graph processing component, GraphX, supports graph parallelism with its vertex-centric API. BFS excels at uniform level-wise exploration but may struggle with big graphs with uneven features. Apache Spark GraphX scales but has a higher learning curve and resource-intensive operations. This comparison helps determine BFS's appropriateness and limits in Apache Spark GraphX's Big Data environment. In example graph data, Figure 5 shows the number of connections each node has. Each bar represents a node, with its height indicating its connections. This visualization shows the graph's connectedness and structure, revealing which nodes have the greatest connections and may be hubs. Nodes 2 and 4 have the most connections, reflecting their centrality in the network.

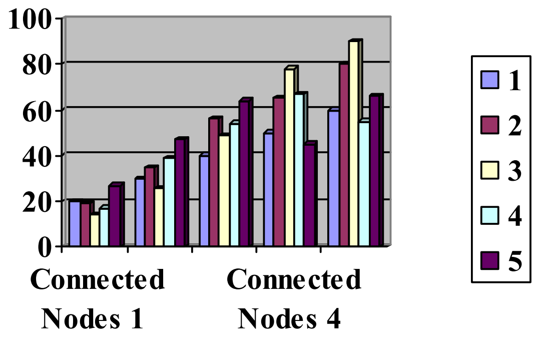

Figure 5: Node Connections in the Sample Graph Dataset

Figure 6 shows Breadth-First Search (BFS) distances between nodes and a source node. Nodes are on the y-axis and their distances from the source on the y-axis. This image shows how distant each node is from the beginning point, which helps explain the graph traversal and each node's level. Nodes further from the source, such node 10, have the maximum distance value, suggesting later BFS stages.

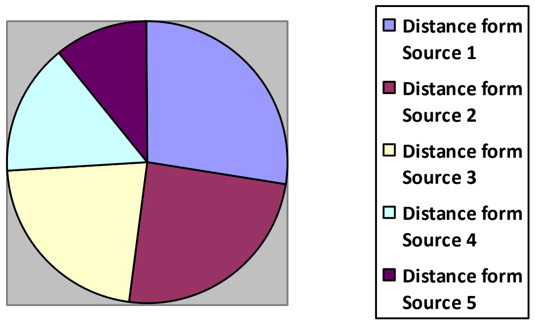

Figure 6: Node Distance from Source

Table 3 shows the problems, effect, limits, and future of BFS in Apache Spark GraphX for Big Data. BFS speed and scalability in large data contexts depend on algorithm design, scalability to handle enormous datasets, fault tolerance, resource management, and optimal graph representation. Improved algorithms, distributed computing, fault tolerance, resource management, and optimized data structures will unlock BFS's full potential for big data analytics, enabling more efficient and scalable graph processing solutions.

Table 3: Challenges And Future Scope of Implementing Breadth-First Search In Big Data With Apache Spark GraphX

Aspect	Challenges	Impact	Limitations	Future Scope
Algorithm Efficiency	Handling large-scale graph traversal efficiently	Influences processing time and resource utilization	May lead to increased computation costs and slower execution	Develop more efficient BFS variants for big data processing
Scalability	Adapting to massive datasets and distributed computing	Determines the system's ability to handle growing data volumes	Inadequate scalability may hinder performance and limit dataset size	Explore distributed BFS optimization techniques and algorithms
Fault Tolerance	Handling node failures and ensuring reliable processing	Affects the system's resilience and fault recovery capabilities	Lack of fault tolerance may result in incomplete or erroneous computations	Enhance fault tolerance mechanisms for distributed BFS
Resource Management	Optimizing resource utilization in distributed systems	Impacts overall system performance and efficiency	Inefficient resource management may lead to underutilization or bottlenecks	Develop better resource allocation strategies for BFS processing
Graph Representation	Efficient storage and manipulation of graph data	Directly impacts BFS performance and scalability	Inefficient graph representation may hinder processing speed and scalability	Explore optimized graph data structures for BFS processing

IV. CONCLUSION

Managing communication overhead and maintaining load balancing across distributed systems are two of the many obstacles to implementing BFS in Big Data with Apache Spark GraphX. The inefficiency in memory use and processing time, particularly with very big and dense graphs, is a result of the complexity of graph data. Algorithm optimization for performance improvement and the development of sophisticated approaches for improved resource management and fault tolerance are both within the purview of future work. Spark GraphX's processing power and flexibility might be further enhanced by integrating machine learning models with BFS. Offering a strong solution for fast graph traversal and analysis, this approach has a substantial influence on big data analytics. Innovation and insights in data-intensive applications may be driven by its ability to handle complicated queries on enormous datasets with enhanced speed and accuracy. This has the potential to revolutionize disciplines including recommendation systems, social network analysis, and bioinformatics. In one example of Node Distance from Source, the numbers vary from 32 to 93 kilometers, while in another case with 5 nodes linked, the results range from 14 to 78 kilometers. When comparing the two sets of data, the values are 23 to 98 kilometers. These ranges are all derived from the Sample Graph Data.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

REFERENCES

[1]. L. Meng, Y. Shao, L. Yuan, L. Lai, P. Cheng, X. Li, W. Yu, W. Zhang, X. Lin, and J. Zhou, “A Survey of Distributed Graph Algorithms on Massive Graphs,” arXiv preprint arXiv: 2404.06037, pp. 1-35, 2024.
[2]. G. Theodorakis, J. Clarkson, and J. Webber, “Aion: Efficient Temporal Graph Data Management,” International Conference on Extending Database Technology, pp. 501-514, 2024.
[3]. S. U. Rehman, M. I. Khalil, M. Kundi, and T. AlSaedi, “A Study on Frequent Subgraph Mining Approaches: Challenges and Future Directions,” Research Updates in Mathematics and Computer Science, vol. 4, pp. 33-63, 2024.
[4]. L. Zeng, H. Huang, B. Zheng, K. Yang, S. Shao, J. Zhou, J. Xie, R. Zhao, and X. Chen, “WindGP: Efficient Graph Partitioning on Heterogenous Machines,” arXiv preprint arXiv: 2403.00331, pp. 1-19, 2024.

[5]. S. B. Gandreti, and S. PS, "Breadth-First Search Approach for Mining Serial Episodes with Simultaneous Events," Joint International Conference on Data Science and Management of Data, pp. 36-44, 2024.

[6]. K. Olgu, T. Kenter, J. Nunez-Yanez, and S. Mcintosh-Smith, "Optimisation and Evaluation of Breadth First Search with oneAPI/SYCL on Intel FPGAs: from Describing Algorithms to Describing Architectures," International Workshop on OpenCL and SYCL, pp. 1-11, 2024.

[7]. M. D. Pratama, R. Abdillah, D. Herumurti, and S. C. Hidayati, "Algorithmic Advancements in Heuristic Search for Enhanced Sudoku Puzzle Solving Across Difficulty Levels," Building of Informatics, Technology and Science (BITS), vol. 5, no. 4, pp. 659-671, 2024.

[8]. P. Gepner, B. Kocot, M. Paprzycki, M. Ganzha, L. Moroz, and T. Olas, "Performance Evaluation of Parallel Graphs Algorithms Utilizing Graphcore IPU," Electronics, vol. 13, no. 11, pp. 1-15, 2024

[9]. S. Lemons, W. Ruml, R. Holte, and C. L. López, "Rectangle Search: An Anytime Beam Search," InProceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 18, pp. 20751-20758, 2024

[10]. Salem N, Haneya H, Balbaid H, Asrar M. "Exploring the Maze: A Comparative Study of Path Finding Algorithms for PAC-Man Game," Effat University, pp. 1-7, 2024.

[11]. J. Zhang, "Distributed Graph Algorithms: From Local Data to Global Solutions," Science and Technology of Engineering, Chemistry and Environmental Protection, vol. 1, no. 5, pp. 1-4, 2024.

[12]. D. Li, B. Yang, L. Liu, C. Chen, C. Sun, L. Ma, S. Xiao, and J. Sun, "Research on a Unified Data Model for Power Grids and Communication Networks Based on Graph Databases," Electronics, vol. 13, no. 11, pp. 1-14, 2024.

[13]. X. Dong, Y. Gu, Y. Sun, and L. Wang, "PASGAL: Parallel And Scalable Graph Algorithm Library," arXiv preprint arXiv:2404.17101, pp. 1-7, 2024.

[14]. Q. Ci, "Research on the Application of Computer Big Data Technology in Smart Travel," Journal of Combinatorial Mathematics and Combinatorial Computing, vol. 119, pp. 291-303, 2024.

[15]. S. Yang, B. Zhao, and C. Xie, "AQA-Bench: An Interactive Benchmark for Evaluating LLMs' Sequential Reasoning Ability," arXiv preprint arXiv: 2402.09404, pp. 1-18, 2024.

[16]. O. Kahramanogulları, "Deep Inference in Proof Search: The Need for Shallow Inference," InProceedings of 25th Conference on Logic for Pro, vol. 100, pp. 370-389, 2024.

[17]. W. H. Li, and Y. C. Huang, "Syndrome-based Fusion Rules in Heterogeneous Distributed Quickest Change Detection," arXiv preprint arXiv: 2405.06933, pp. 1-7, 2024.

[18]. I. Ben-Noah, J. J. Hidalgo, and M. Dentz, "Note on Using Singular Value Decomposition to Solve Diluted Pore Networks," arXiv preprint arXiv:2403.13462, pp. 1-16, 2024

[19]. S. Climer, K. Smith Jr, W. Yang, L. D. Fuentes, V. G. Dávila-Román, and C. C. Gu, "Sifting out communities in large sparse networks," arXiv preprint arXiv: 2405.00816, pp. 1-24, 2024.

[20]. V. Chelkovan, "The Use of Artificial Intelligence Methods in the Intelligent Decision Support System," Electronics and Control Systems, vol. 1, no. 79, pp. 16-21, 2024.