technologies

Article

Benchmarking Big Data Systems: Performance and Decision-Making Implications in Emerging Technologies

Leonidas Theodorakopoulos [1], Aristeidis Karras [2,*], Alexandra Theodoropoulou [1] and Georgios Kampiotis [1]

[1] Department of Management Science and Technology, University of Patras, 26334 Patras, Greece; theodleo@upatras.gr (L.T.); theodoropouloua@upatras.gr (A.T.); up1059197@ac.upatras.gr (G.K.)
[2] Computer Engineering and Informatics Department, University of Patras, 26504 Patras, Greece
* Correspondence: akarras@ceid.upatras.gr

Abstract: Systems for graph processing are a key enabler for insights from large-scale graphs that are critical to many new advanced technologies such as Artificial Intelligence, Internet of Things, and blockchain. In this study, we benchmark another two widely utilized graph processing systems, Apache Spark GraphX and Apache Fink, concerning the key performance criterion by means of response time, scalability, and computational complexity. We demonstrate our results which show the capability of each system for real-world graph applications, and hence, providing a quantitative understanding to select the system for our purpose. GraphX's strength was in processing batch in-memory workloads typical of blockchain and machine learning model optimization, while Flink excelled in processing stream data, which is timely and important to the IoT world. These performance characteristics emphasize how the capabilities of graph processing systems can match the requirements for the performance of different emerging technology applications. Our findings ultimately inform practitioners about system efficiencies and limitations, but also the recent advances in hardware accelerators and algorithmic improvements aimed at shaping the new graph processing frontier in diverse technology domains.

Keywords: graph processing systems; Apache Spark GraphX; Apache Flink; performance benchmarking; decision making; emerging technologies; AI; IoT; blockchain; big data

check for updates

Citation: Theodorakopoulos, L.; Karras, A.; Theodoropoulou, A.; Kampiotis, G. Benchmarking Big Data Systems: Performance and Decision-Making Implications in Emerging Technologies. *Technologies* **2024**, *12*, 217. https://doi.org/10.3390/technologies12110217

Academic Editor: Valeri Mladenov

Received: 19 October 2024
Revised: 30 October 2024
Accepted: 30 October 2024
Published: 3 November 2024

1. Introduction

Originally, big data arose from different entities like social media, IoT devices, and digital transactions which require strong tools to analyze massive and complicated data. Among the Transactional Data Systems (TDSs), graph processing systems, in particular, can be pointed out as important means for analyzing big graph datasets that occur in relationships and structures in the data. These systems are now commonly applied in social network analysis platforms, recommendation systems, biological projects, and logistics.

However, the rapid development of novel technologies such as Artificial Intelligence (AI), the Internet of Things (IoT), and Blockchain has created new needs and challenges for graph processing systems. These technologies produce massive and intricate data over time, and the required systems should be elastic and capable of providing a rapid output. Here, the performance matters most in the decision-customizing stages caused by data processing, which determines the effectiveness of the decision made, as in AI for predicting outcomes promptly, IoT network for distributing resources in real-time, or Blockchain for verifying transactions.

Due to the vast number of graph processing systems present in the current computing environment, it is, however, not easy to determine the most suitable system to use for a given use case scenario. Impact plays a major role in changes in performance based on several factors such as response time, system capacity, and resource consumption, depending on a specific system. Lastly, performance comparison can be a very tiring and tiresome process that requires some guidelines and parameters that could make the comparison true and fair.

This study focuses on comparing two popular distributed graph processing frameworks: Apache Spark GraphX and Apache Flink. Both systems are popular, since they handle big graph data in distributed ways at different stages; however, they have different structures and operation modes. Apache Spark GraphX is a graph computational system developed on the Spark platform that transparently defines complex computation of graphs, one of the most complex data systems with excellent in-memory data handling optimization. In contrast, Apache Flink is preferred for real-time stream processing, which can be the major reason for choosing it for IoT applications.

The performance comparison of this study focuses on Apache Spark GraphX and Apache Flink based on a set of graph processing queries that include the shortest path, degree centrality, triangle count, and number of weakly connected components. To make this comparison fair, we compare these systems based on the same datasets, workload, and hardware environments to establish areas where one can overpower the other. The intended outcome is to facilitate an understanding of how the best between the systems can be arrived at to fit different real-world applications.

Besides the comparison between the performance of the two groups of algorithms, this paper also examines the applicability of the findings to decision-making in emerging technology settings. For instance, the use of artificial intelligence in systems demands quick data processing of small-world network models, or on the other hand, analyzes the graph-based in the IoT technology, which is vital in device interaction and resource control. In BM-SRC, graph processing is used in the transaction verification process and in consensus algorithms, where efficiency has direct implications for safety and capacity.

To make sure we make our evaluation as complete as possible, we will employ a system capable of running predefined queries on each of the systems under test, timing how long each of those tasks takes, and then calculating various performance measures. We will also generate figures of each of the above metrics for each of the systems for comparison at a glance from the figures. These opportunities prove that our methodology is fair because all of the systems under test are trained, tested, and evaluated on the same dataset, receive identical processing workloads, and are tested on the same hardware platform. The findings will be checked on various parameters to confirm their credibility. Ultimately, this study evaluates the performance of Apache Spark GraphX and Apache Flink but also presents a focus on the implication of graph processing performance in decision-making in the context of emerging technologies. The insights will help researchers and practitioners identify the right graph-processing system for their context in accordance with their applications and technological settings.

The remainder of this study is arranged as follows. In Section 2, we discuss the background and related work by analyzing existing graph processing system and their relationship to new technologies. Section 3 presents the experiments' design, focusing on the system configuration, data acquisition, and execution of the graph processing systems. Section 5 provides a comparison between Apache Spark GraphX and Apache Flink based on execution time and memory usage. Finally, the results are discussed in Section 6 providing information on decision-making concerning the comparison of the two systems. Finally, the conclusions and directions for future work are presented in Section 7.

1.1. Aim of the Study

The main objective of this research is, therefore, to measure and compare the effectiveness of two of the most widely used graph processing frameworks, namely Apache Spark GraphX and Apache Flink, for big data environments. More generally, this research aims to compare these frameworks by using graph-based algorithms including degree centrality, shortest path, and triangle counting on different large-scale datasets. The paper also discusses the impact of graph processing efficiency on business decisions in novel areas of advancements like AI, IoT, and blockchain. Consequently, the results provide an understanding of the benefits and drawbacks of these systems regarding their ability to scale, the time needed to execute the algorithm, and the number of resources used to achieve this and help the stakeholders to choose the most suitable framework for the particular case.

1.2. Limitations

This work has conducted an extensive comparison between Apache Spark GraphX and Apache Flink. Firstly, the evaluation is restricted to a small set of graph algorithms that mostly cover fundamental workloads, such as degree centrality, shortest path, and triangle counting. The advanced algorithms, or those specific to an application, like machine learning algorithms, dynamic graph algorithms, etc., are not included in this section. Secondly, the experiments are carried out in setup configuration to the actual system environment and real variation in the arrival of hardware, network architecture as well as in Workload. Lastly, the research is limited to snapshot analysis and does not consider the effectiveness of these systems on real-time evolving graphs, which is becoming a popular field of study in real-time data processing.

1.3. Research Gap

The current study points out that there are limited studies performed on the benchmarking of distributed graph processing systems to different kinds of applications, specifically in real-time and batch-style processes. For example, Guo et al. highlighted the need for a more mature analysis of graph processing platforms' performance through a lack of comprehensive benchmarking suites and a limited understanding of the impacts of dataset and algorithm diversity on performance [1]. Dayarathna et al. [2] also demonstrated that benchmarking graph stream processing and graph-based machine learning suffers from gaps, and more robust benchmarks that can handle high workload scenarios are needed. In addition, Uta et al. mentioned elasticity-related challenges in graph processing that are still underexplored in the benchmarking studies [3].

Present day, Apache Spark GraphX and Apache Flink are normally utilized in big data processing; however, there is not enough genuine comparison of their continual performance for changing datasets and under differing loads. The scalability and efficiency of Spark GraphX in handling large-scale graph data are well-known in applications at Tencent [4] and bioinformatics [5]. In addition, it is used for semantic web data processing and financial fraud detection proving its versatility in big data environments [6,7].

On the other hand, Apache Flink is popular for running in real-time data applications, due to its ability to serve as a batching as well as a streaming processing engine [8]. Performance on stream processing is praised, and it has been used in several contemporary research and industrial applications [9,10]. Processing big data with Flink is further improved by its multi-query optimization and dynamic data partitioning strategies.

Additionally, there is a deficiency in the existing literature attempting to explain how AI/IoT and other developing technologies could benefit from improvements in graph processing systems. Notably, future studies should address the improved optimization algorithms for such systems; future research should focus on current optimization techniques as well as usage of the current GNNs and another contemporary algorithm in order to boost the Graph Process capability in a practical environment.

2. Background and Related Work

2.1. Evolution of Graph Processing Systems

The graph processing systems landscape subsequently evolved from single-node architectures to sophisticated distributed frameworks that can handle massive and dynamic graph data. Gonzales et al. introduce the development of GraphX, which uses a distributed data flow system for performance comparable to specialized graph systems [11]. As suggested by Zhuo et al., SympleGraph is a distributed framework that extends graph processing to achieve precise loop-carried dependencies, illustrating the trend of using increasingly elaborate distributed architectures [12]. Heidari et al. presented a taxonomy of scalable graph processing frameworks to facilitate the passing from single nodes to distributed systems for efficient processing of large-scale graph data [13]. The progression to this solution has been fueled by the growing complexity and scale of graph datasets used in diverse domains which require fast and scalable processing solutions. In this subsection, we present key studies that discuss important achievements and innovations in the development of graph processing systems.

The integration of Graph Neural Networks with traditional graph processing systems is a pivotal development in this field. Vatter et al. discussed the maturing of GNNs and their symbiosis with the ways of learning that deep models have come to practice. This survey points out the necessity of scalable GNN training solutions to handle large-scale graphs, in particular bridging the gap between conventional graph analytics and advanced machine learning techniques [14]. On the other hand, with the advent of quantum computing, there appeared new paradigms in processing graphs. The study conducted by Henry et al. introduced a quantum approach toward machine learning on graphs by using quantum processing units to compute graphs directly. The enhanced computational opportunities for machine learning on graphs that this method enables can potentially provide exponential speed up of some graph algorithms [15].

The model proposed by Daverio et al. addresses the dynamic nature of real-world graph data. Capturing temporal dependencies is important for optimizing the processing of evolving graphs and it is one important application in, for example, social network analysis and communication networks, where graph structures evolve [16].

Graph processing systems have also seen wide-ranging hardware innovations in their evolution. Tesseract is a processing-in-memory accelerator described in this study [17], which improves the performance and energy efficiency of graph processing tasks on a large scale. Tesseract can overcome the memory bandwidth bottleneck that limits the performance of many graph algorithms by bringing computation closer to memory, as in Tesseract Accelerator [17].

Bhattacharya et al. utilized software engineering to understand the software evolution by graph topology analysis. The modeling of software components and their interactions as graphs is useful in this study offering some software development and maintenance processes insights [18].

Moreover, graph processing is unique within cloud computing environments, and presents both opportunities and challenges. The research conducted by Malicevic et al. evaluated graph processing systems such as X-Stream in cloud settings from the perspective of challenges and solutions. The network bandwidth was identified as being a performance bottleneck, and solutions were addressed that attempt to optimize graph processing within cloud infrastructures to facilitate more efficient graph processing of large-scale graphs in distributed environments [19].

To combat structural changes in massive graphs, dynamic graph processing systems have emerged. Vaquero et al. proposed a Dynamic Graph Processing System that repartitions huge graphs to adapt to evolving graph structures. By improving computational performance at the cost of data replication, this approach is computationally efficient and can be used for real-time applications where the graph topology itself is changing quite often [20]. GNNs are still evolving to leverage current trends of integrating GNNs with traditional graph processing systems improving the line between deep learning and graph analytics.

Overall, this represents a step change in graph processing moving from single-node systems to distributed frameworks and specialized hardware accelerators. These advances permit the analysis of large, complex, and dynamic graphs which are important for modern applications, e.g., machine learning, software engineering, and cloud computing. Research and development in this area will be necessary for the supporting needs of big data analytics and graph-based data to achieve their full potential.

Emerging Technologies and Graph Processing

Graph processing systems for large-scale data analysis are increasingly relied upon by technologies like the IoT, AI, and others. The data produced by these technologies are vast amounts of interconnected data, and graph processing systems are well suited to process and analyze these data [21]. For example, AI in smart cities can use graph processing to process data from several sources (sensors, cameras, and social media) to ensure that decision-making and operational effectiveness are improved [22]. Like IoT systems, graph processing is also used within IoT systems to process incoming data from numerous connected devices to perform real-time monitoring, predictive maintenance, and resource optimization [23]. Since graph processing is efficient in the deployment and analysis of transaction networks, blockchain—with its decentralized structure—can benefit significantly from it in both ways [24].

Moreover, these graph processing systems are used where organizations draw upon performance metrics of the graph processing systems, like the query execution time or the scalability measurement, to make strategic decisions in these applications [25]. The metrics show how efficient and robust graph processing platforms are under study and can be chosen and tuned based on organizational needs [26]. Performance data can show if a certain system is good for real-time analysis on IoT applications or if it is suitable for higher throughput on a blockchain network. Knowing these performance characteristics helps organizations match their technological capabilities to business objectives, driving better operations and decision-making throughout the AI, IoT, and Blockchain ecosystems.

2.2. Decision-Making Framework

In our analysis, we concentrate on a decision-making framework specific to the unique demands of graph processing in big data environments, which includes three primary factors:

1. Performance Metrics in System Selection: For applications of real-time analytics and batch processing, the choice of graph processing systems is critical to key metrics such as execution time, scalability, and resource efficiency. All the examples above can be extended as low latency and resource-efficient systems are generally indicated for IoT, but batch processing for example which is needed in retrospective data analysis can leverage memory capabilities.
2. Domain-Specific Requirements: The functional requirements vary significantly based on the domain (AI-driven applications, IoT networks, decentralized systems in blockchain) in which the solutions will be used or deployed. We use our framework to select by matching System capabilities (such as real-time processing with Flink and in-memory batch processing with GraphX) to these different requirements.
3. Scalability and Adaptability in Emerging Technologies: With all these rapid AI, IoT, and blockchain advancements, systems are required to offer dynamic data capabilities and seamlessly integrate with new hardware accelerators. One of these factors is that the systems that can be adopted depending on the changing workloads can scale or change without degradation to performance.

This decision-making framework aspires to narrow the scope to deliverable selection criteria for big data applications that can help in decision making by selecting performance, domain fit, and adaptability as the core aspects.

Graph Processing Systems in Decision Making

In real-time data processing, anomaly detection, and resource allocation in AI systems and IoT devices, decision-makers need to evaluate different performance metrics in frameworks like Apache Spark GraphX and Apache Flink.

Depending on use cases, the choice of framework can drastically affect how quickly and effectively applications can make use of it.

- Apache Spark GraphX is well known for very fast in-memory processing, so you can think of the high-speed data handling and fault tolerance for use cases where you need to access data quickly and do iterative processing [11].
- In contrast, Apache Flink shines in stream processing with low latency and high throughput and is a perfect option for real-time data stream processing in IoT environments [27].

Key Performance Metrics:

1. Execution Time: The data processing speed that a framework gets is the basis of this metric. Consider Apache Spark, which is known to be much faster than Flink when there is enough memory to execute iterative tasks [28,29], or Flink specifically for low latency scenarios which are good for real-time applications.
2. Scalability: The framework scalability stands for its capability to keep running with growing volumes of data without a performance drop. While Apache Flink is often preferred as it may scale well in stream processing environments [29,30], Spark might do better for batch processing tasks.
3. Resource Utilization: The evaluation of how well we are utilizing today's computational resources. Spark can also take advantage of the capabilities of the graphics processing unit (GPU) using its GraphX API, which will speed up graph analytics workloads [31]. Compared to Flink, its architecture enables a dynamic allocation of the resource during stream processing. Furthermore, continuous data streams in IoT prefer Flink, as it has efficient resource management [32].
4. Fault Tolerance: Frameworks offer support for fault tolerance but do it differently. This is particularly critical if the processing happens in a real-time scenario, as with Flink, which uses a checkpointing mechanism for recovering from failures seamlessly [33].

2.3. Technological Advancements in Graph Processing Systems

Over the past few years, tremendous progress has been made in graph processing systems as the need to efficiently manage and analyze large-scale graph data has grown. Such advancements are unprecedented and pivotal in many other areas of business, in particular smart cities, industrial automation, and information technology, where complex data relationships are inherent.

An interesting aspect of development is the integration of graph processing in smart city infrastructures. Analyzing traffic has been explored in the context of transportation management using Apache GraphX for real-time usage. By following this approach, system throughput and processing time have improved substantially, enabling more responsive and efficient urban traffic solutions [34].

GPU-enabled graph processing systems have also advanced high-performance computing. Systems like Medusa and Totem have been subject to an empirical evaluation that has revealed insights into their performance and scalability. Graph computations in these systems take advantage of their parallel processing capabilities offered by GPUs to reduce the time taken for large-scale graph analytics [26].

With Knowledge Graphs and Semantic Web technologies, the fusion of the industrial domain goes forward into Industry 4.0. Researchers have improved asset monitoring and process optimization by proposing an enriched ontological model. This integration makes way for smarter industrial operations by allowing for a deeper semantic understanding of exchanged industrial data as the integration enables better data interconnectivity [35].

GraphScope Flex exemplifies the advancements in modularity and flexibility of the graph computing stacks. This LEGO-style architecture promotes the modularity of design, paving the web for a customizable and efficient graph computing solution. Additional performance improvements are also gained by allowing components to be assembled depending on specific computational needs [36].

As graph processing systems, they have also helped improve cybersecurity measures. Improved control over prediction and classification tasks is achieved through the application of graph signal processing algorithms for anomaly detection in IT infrastructures. Additionally, it improves the pace of identifying and addressing security threats [37].

To improve version switching speed and memory efficiency, systems such as Version Traveler have been designed to address the challenges of multi-version graph data. Given the need for rapid access to multiple versions of graph data while minimizing memory overhead [38], this innovation is key to such applications.

Graph partitioning remains an important problem in terms of balancing the load and minimizing communication overhead for real-time applications. To improve performance in distributed graph processing environments, application-aware adaptive partitioning strategies have been proposed that dynamically vary the partitions based on the characteristics of the application [39].

In the context of hardware architecture, the development of HNGraph is finely tuned for improving the performance of Non-Uniform Memory Access (NUMA) systems. HNGraph optimizes memory access strategies for hybrid memory-based NUMA systems to support parallel graph processing more by utilizing the underlying hardware, resulting in improved computational efficiency [40].

Collectively, these graph processing system improvements are imperative to contend with the rising volume and complexity of graph data. For an increasingly data-driven world, they provide innovative scalability, efficiency, and adaptability solutions that are critical to the development of data analytics and artificial intelligence applications.

2.4. Related Work on Apache Spark GraphX and Apache Flink

Two leading frameworks in big data analytics and graph processing are Apache Spark GraphX and Apache Flink. There has been a lot of work to compare their performance, seamless degree of scalability, usability, and application in different domains. In this subsection, we review key studies that contribute to our understanding of the strengths and limitations of both of these frameworks.

Marcu et al. analyzed the performance of Spark and Flink and evaluate their performance concerning the architectural differences and varying configurations of parameters. In general, neither Spark nor Flink framework outperforms the other over all the categorizations of data types and workloads, indicating that compared to the other option, Spark or Flink is highly data-type- and workload-specific [41].

These frameworks have been tried and tested in the healthcare industry, an industry filled with sensitive data at its disposal. Nazari et al. compared Apache Hadoop, Apache Spark, and Apache Flink in terms of healthcare data analysis. They are strong on processing speed and fault tolerance which are both necessary for handling large-scale healthcare datasets, the research says. It shows that though both frameworks are suitable, the preferred one depends on the distinct demands of the healthcare applications [8].

Data scientists choose a big data framework primarily for usability. Akil et al. reviewed user preferences and also uncover no considerable contrast between Spark and Flink for immense information preparation as far as read and compose times. This implies that the decision-making process may be influenced by other factors apart from the evaluation of the usability within the particular tool, for example, how the community supports a tool or how it is integrated into the existing suite of tools [42].

For machine learning problems performance and scalability are critical. Garcial-Gil et al. showed that Spark's MLlib usually achieves better performance and lower runtimes than Apache Flink. In this respect the advantage that Spark offers it is better suited to batch processing machine learning workloads that require efficiency, as has been proven previously [29].

Real-time data analytics demands stream processing capabilities. Karakaya et al. compared Flink, Spark, and Storm and conclude that under equal constraints, Flink outruns the other two in stream processing scenarios. While the study does show that Spark can be optimized for higher throughput, Spark may be optimized so that its performance can be dramatically improved.

In [43], Chen et al. explored the impact of abstraction layers on performance. Performance penalties imposed when Apache Beam is used as an abstraction layer over Spark and Flink are considered in this research. This study reveals significant performance degradation and shows there are trade-offs between ease of development and system efficiency.

Blamey et al. compared Spark Streaming with HarmonicIO in A Performance and Architecture Comparison. To provide insights into their performance under different message sizes and CPU loads, this study helps better understand how Spark Streaming copes with varying operational conditions [44].

Boden et al. compared the scalability of Spark and Flink for machine learning algorithms. The shortcomings of these two frameworks in handling high-dimensional data are a point of research showing that neither one is optimized for machine learning tasks with many features. This indicates therein is a potential need for further optimization and ultimately for the existence of application-specific tools in such scenarios [45].

Ultimately, we have attempted to compare and contrast Apache Spark GraphX and Apache Flink, and the conclusions regarding the comparative analyses of Apache Spark GraphX and Apache Flink are summarized below. In many cases, this choice is between the two depending on the application's specific needs (workload type, performance requirements, or specific domain considerations). Practitioners need to make informed decisions and big data processing systems are constantly improved in ongoing research and performance benchmarking.

2.5. Anomaly Detection and Financial Transaction Pattern Analysis in the Accounting Sector

Accounting is becoming more and more reliant on sophisticated anomaly detection and financial transaction pattern recognition to enhance fraud detection, compliance, and operational efficiency. In the accounting and finance field, Bhowte et al. provided a systematic literature review about the machine learning techniques for detecting efficiently fraudulent activities and for fraud detection applications [46]. In addition, Thilagavathi et al. utilized Graph Neural Networks combined with anomaly detection to increase fraud detection in financial transactions and prove its significant improvement compared to traditional methods [47].

With the exponential growth of digital transactions, the importance of Artificial Intelligence (AI) and Machine Learning (ML) in securing financial transactions is critical. The first, and probably most apparent, is that one of the major methods of fraud detection uses AI-based systems that track financial transactions as they occur, and determine what is unusual. According to an extensive review of the literature from 2010 to 2023, these systems have greatly improved and are very successful in catching fraudulent activity at the beginning of a transaction. These AI systems use real-time anomaly detection, which makes electronic payments much more secure because they are constantly learning from transaction data and adapting to new fraud patterns. Consequently, this approach has become a cornerstone for corporations and financial institutions aiming to strengthen their fraud prevention strategies [48].

Machine learning has also been very helpful in analyzing large financial data streams and is good at recognizing repetitive transactional patterns in the fluid world of today's financial systems [49–51]. Different research emphasizes the power of ML algorithms to handle and identify patterns in large, streaming data. This ability is vital to accounting firms because decisions must be made at the moment and they are only as good as their interpretation of the pattern of financial transactions. Furthermore, not only that, but strides in ML technology have allowed for the development of adaptive systems which cater to the specific needs of unstable data streams, providing for much more accurate transaction tracking and regulatory checking [52].

Furthermore, there is also network analysis, which is used to look at patterns of financial transactions, such as in a paper on the financial networks of Austria. This paper investigates how the topology of the network changes at different levels of observation time and shows a low correlation between the degree of nodes and the volume of transactions, implying that more subtle metrics are needed to fully grasp the dynamics of transactions. Using eigenvalue analysis, they can better understand the dependencies of the network and the trading patterns of the major financial players [53].

Artificial Neural Networks (ANNs), especially the back-propagation (BP) model, have been applied to the study of blockchain financial transactions. One paper is about transaction propagation, weight changes, learning mechanisms, and its application to fraud detection in the blockchain. Its applications, like this neural network, are so priceless in tracking decentralized transactions that other means are worthless [54].

In addition, when dealing with high-frequency financial data, one can use the combination of the theory of marked point processes and extreme value analysis to get accurate measures of tail dependencies in the financial transaction data [55]. This method can model the correlation between transaction times and extreme return behaviors, and therefore, give much more precise risk estimates. For this purpose, econometric tools are also essential due to their specificity toward transaction data challenges, including timing and volume, uniquely faced in transaction data rather than in other data types [56]. These different techniques are highlighted in a broader survey of econometric methods, which completes a survey of econometric methods relevant to the analysis of financial transactions [57].

Similar to the studies mentioned above, some studies use social network analysis and transaction flow analysis, complying with Financial Action Task Force (FATF) guidelines as a technique to fight money laundering [58]. These systems will effectively serve as systems for monitoring compliance and detecting financial crimes by mapping the transaction flows and detecting suspicious patterns. In addition, models such as the weighted one-class support vector machine (WOC-SVM) have also been effective in finding atypical transactions, making it useful for financial institutions to stay in regulatory regime compliance [59].

In an innovative study, researchers have applied the Multiple Benford Law Model and K-Means Cluster to improve the accuracy of fraud detection. Using clustering techniques with statistical rigor, this method reached a 93.33% detection rate and provides a very effective auditing tool for auditors and financial analysts trying to uncover fraud [60].

Finally, all these various approaches simultaneously underline the work's complex and diverse nature concerning anomaly detection and pattern recognition in the accounting field. This evolution of currency indicates that the approaches to detection and protection of financial transactions should follow this pattern and pave the way for better, smarter systems.

Financial transaction analysis is becoming an increasingly important area for anomaly detection as it pertains to real-time insight, fraud prevention, and strict regulatory compliance. Institutions use graph processing systems like Apache Spark GraphX and Apache Flink to pinpoint complex transaction patterns and preventively tackle possible risks. More specifically, metrics such as degree centrality, shortest path analysis, and triangle counting provide the granular analysis of transactional behaviors, accounting for accounts, flows, or cycles that may portend irregular activities. The capacity to discern anomalies with fine specificity falls completely in line with the study's goal of assisting with decision making in high-stakes arenas.

These metrics can then be incorporated into a decision process framework to aid organizations in selecting the best system for a required task. For instance, degree centrality reports accounts with ultra-active transaction volumes for high-risk entity detection in real time. Shortest path analysis provides the means for focusing on direct and indirect fund flow between accounts necessary to find the quick fund transfers that typify fraud. A key component of detecting coordinated behaviors such as money laundering cycles is triangle counting which determines cyclic transactions.

In combination with the decision-making framework, this study allows the analytical metrics to be integrated enabling clear criteria of system selection based on performance, resource efficiency, and scalability. The strengths of each system, i.e., the technical capability of GraphX for batch processing and Flink for real-time analysis, are assessed not only in terms of their technical capability but also in terms of their implementation in safeguarding financial integrity. Thus, the results are useful for actioning as actionable insights for financial institutions in optimizing system choice in terms of the strategic objectives of anomaly detection and regulatory compliance.

3. Methodology

The methodology for this experiment, as illustrated in Figure 1. In particular, Figure 1 follows a structured approach to compare the performance of two highly used graph processing systems: Apache Spark GraphX and Apache Flink. Data acquisition and pre-processing undertake cleaning and structuring of raw data and help prepare the data for analysis. Once the system is set up and configured, we execute core graph processing algorithms, like Degree Centrality, Shortest Path Calculation, Triangle Counting, and Weakly Connected Components. Execution of these algorithms is performed using Apache Spark GraphX and Apache Flink. Execution time, memory usage, and scalability are then collected and analyzed with dissimilar performance metrics. An analysis of the results of this analysis offers key insights to direct decision making and select the most appropriate system for given graph processing tasks.

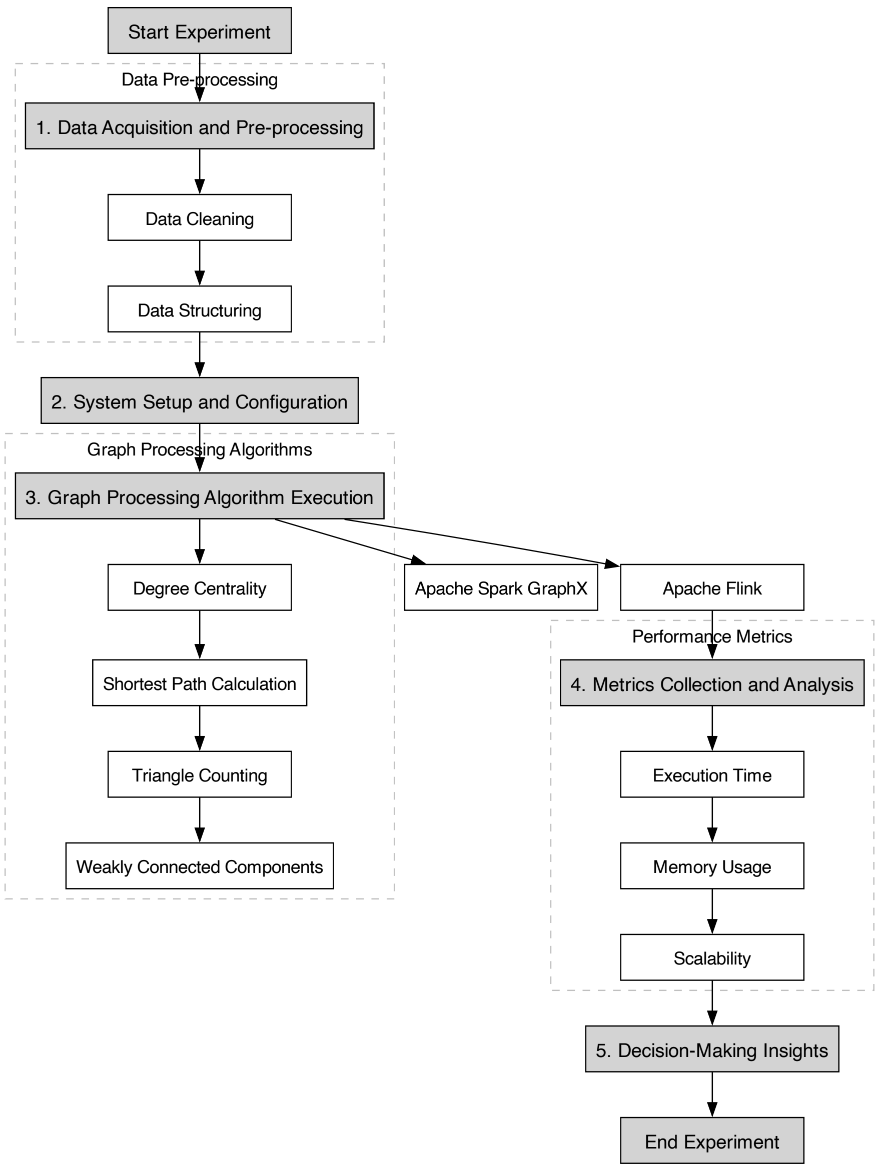
Start Experiment
Data Pre-processing
1. Data Acquisition and Pre-processing
Data Cleaning
Data Structuring
2. System Setup and Configuration
Graph Processing Algorithms
3. Graph Processing Algorithm Execution
Degree Centrality
Shortest Path Calculation
Triangle Counting
Weakly Connected Components
Apache Spark GraphX
Apache Flink
Performance Metrics
4. Metrics Collection and Analysis
Execution Time
Memory Usage
Scalability
5. Decision-Making Insights
End Experiment

Figure 1. Flowchart of the experimental methodology.

The methodology used in the case study, as seen in Figure 2, is of a similar structured approach. Here, transactions and accounts are modeled as nodes and edges in a financial network to find patterns in transactions and accounts that seem suspect and which could constitute anomalies. Using Key graph processing algorithms such as Degree Centrality, Shortest Path Analysis, and Triangle Counting, we define abnormal transaction behaviors. Apache Spark GraphX and Apache Flink demonstrate their suitability for decision making in the financial sector in this case study.

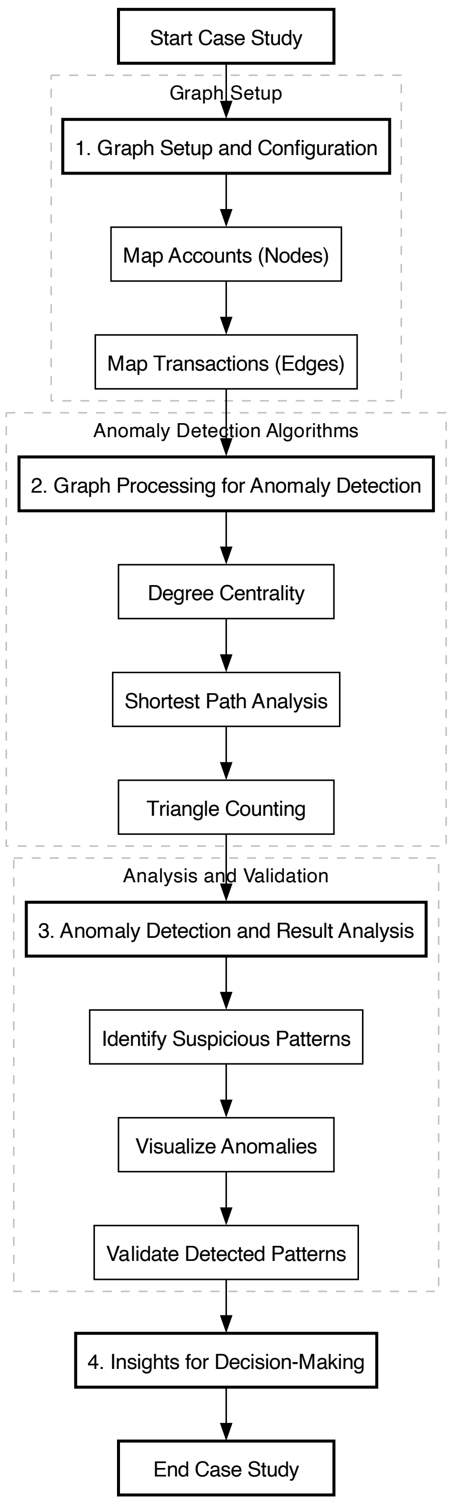
Start Case Study
Graph Setup
1. Graph Setup and Configuration
Map Accounts (Nodes)
Map Transactions (Edges)
Anomaly Detection Algorithms
2. Graph Processing for Anomaly Detection
Degree Centrality
Shortest Path Analysis
Triangle Counting
Analysis and Validation
3. Anomaly Detection and Result Analysis
Identify Suspicious Patterns
Visualize Anomalies
Validate Detected Patterns
4. Insights for Decision-Making
End Case Study

Figure 2. Flowchart of the case study in accounting.

3.1. System Setup

The design for this experiment involved a benchmarking of Apache Spark GraphX and Apache Flink in terms of standardized graph processing tasks on a powerful hardware infrastructure best suited to large data analysis. The environmental setup contains the system characteristics, cluster layout, as well as the software installed, so all the results can be compared on equal grounds.

Hardware Configuration

The experiments were conducted on a single powerful machine with the following specifications, as shown in Table 1. This layout results in the availability of the required computational capabilities to meet the processing requirements of big graph datasets and facilitates performance calibration.

Table 1. System specifications for experimental evaluation.

Component	Specification
CPU	AMD 5950X
RAM	64GB DDR4
Storage	2TB NVMe
GPU	3090Ti (24GB VRAM)
Operating System	Ubuntu 18.04 LTS
Network Interface	1GBPS
Software Framework	Apache Spark with FATE

More than 32 cores in the AMD 5950X processor, enough RAM, and a lightning-fast NVMe drive make it easy to quickly load and process data. Moreover, the NVIDIA 3090Ti GPU has multi-threaded support for parallel processing, which simultaneously improves graph processing tasks that take advantage of GPUs.

3.2. Cluster Configuration

The experiments employed a one-node environment that implements the master and one working node for Apache Spark and Apache Flink using their parallel processing modules. Although there was a plan to use a cluster of several nodes, using a powerful single machine offers a clean environment where resource usage can be measured independently of network overhead.

3.2.1. Apache Spark Configuration

Apache Spark was configured as follows:

- Spark Master: Run on the local machine on the default port 7077, which is set up for this set.
- Executor Cores: Further set to utilize 16 cores which are in tandem with the high number of cores of the CPU.
- Memory Allocation: For proactive processing of graph-related tasks, 8 GB of memory was allocated to each of the executors.
- Driver Memory: Configured at 4 GB to meet the needs for the handling of jobs and communications.

3.2.2. Apache Flink Configuration

Apache Flink was also configured on the same machine with the following settings:

- JobManager: For job monitoring, was operated at port 8081 on its local machine.
- TaskManager Slots: That had been configured to make use of eight task slots and take the APU's parallel processing capabilities full advantage.
- Memory Allocation: The TaskManager was given 16GB to use to handle the workload.
- Parallelism: Match to stack, set to 16, core count which maximizes resource utilization.

4. Software Installation and Setup

The setup involved installing the necessary software components to ensure a stable and consistent environment:

- **Java Development Kit (JDK)**: Apache Spark and Flink are compatible with Java versions 8, 11, and 17. Java 11 is recommended for compatibility with the latest versions of both frameworks.
- **Apache Spark**: Version 3.5.3 was used, which includes the Spark GraphX library for graph processing and provides improved performance and features for SQL and streaming functionalities.
- **Apache Flink**: Version 1.17.1 was configured with the Flink Gelly API, which contains essential functions for graph processing.
- **Scala and SBT**: For compatibility with Spark 3.5.3, Scala 2.13, and SBT 1.8.2 were installed to compile and run Spark jobs efficiently.
- **Maven**: Version 3.8.6 was used for compiling and managing dependencies for Flink jobs, ensuring compatibility with the latest Apache Flink version.

4.1. Experimental Procedure

We designed the experiments to compare Apache Spark GraphX and Apache Flink on degree centrality, shortest path, and triangle counting. Performance metrics (execution time, memory usage, scalability) for each experiment were obtained by running each experiment multiple times. To avoid further fluctuations in the results, the runs were averaged and repeated a few times.

To evaluate the systems, both the Google Web Graph dataset and the Anti-Money Laundering Simulator (AMLSim) dataset were used for the dataset evaluation under different scenarios. We recorded the dataset characteristics and performance metrics for each graph processing task and compared the two systems under realistic conditions.

A systematic approach to system setup and configuration guarantees that each test case can represent Apache Spark GraphX and Apache Flink capabilities faithfully, for informative insight into their suitability to support large-scale graph processing in emerging technology applications.

4.2. Dataset

The selection of the dataset is a significant factor in evaluating performance in graph processing systems. A really good dataset would be something that is of realistic proportions to real-world situations and is large and complex enough to give the systems that are being tested a run for their money in terms of their scalability and efficiency. In this section, we describe the datasets selected for our comparison, emphasizing their characteristics, representativeness, and alignment with the study's objectives.

4.3. Google Web Graph Dataset

The Google web graph dataset is a very popular graph to work with graph processing research [61]. The World Wide Web is the example that this graph is modeling, where the web pages are the nodes and the hyperlinks are the directed edges. As a representative of real web data, it contains 875,713 nodes and 5,105,039 edges, and therefore, it is a good test for the graph processing system's scalability and computational efficiency on real web data.

This dataset was originally released by Google in the Google Programming Contest and later used in several research studies because of its availability and representativeness [61]. We use the Google web graph dataset in this experiment to test the functions of Apache Spark GraphX and Apache Flink in reference to their capacities in processing large graph structures.

These are data from the Stanford Network Analysis Platform, which has a trove of large datasets available for graph processing research. Large and well-connected, it is a good test of the ability of each system to deal with a large, complex network. Measures of statistics such as degree distribution, size of the largest weakly connected component, and triangle count give insight into how the systems handle large, structured web data.

We explore several datasets to evaluate and select the Google web graph dataset due to its compatibility with our computational resources and research goals. The size of the dataset is large enough to support real conclusions about how Apache Spark GraphX and Apache Flink compare in execution time, scaling, and the processing power that is available for solving graph structures found on the web. Such a large corpus fits well to the performance analysis over a predefined web-like load as well as produces a strong basis of comparison for the system's ability.

4.4. AMLSim (Anti-Money Laundering Simulator) Dataset

In order to run applications on the Google web graph dataset, we introduce the AMLSim dataset (Anti-Money Laundering Simulator) for anti-money laundering analysis and exploration of the applicability of graph processing systems on financial problems, namely on finding patterns and abnormalities [62]. The transaction networks in this dataset are modeled as directed edges connecting nodes—bank accounts—representing bank accounts and transactions. With each transaction attributed to the amount, timestamp, and account type, it is a perfect simulation of money laundering in the real world.

This study selected the AMLSim dataset because it is most relevant to anomaly detection tasks and can represent complex transactional relationships within financial networks [62]. This makes it possible to use graph processing algorithms like degree centrality, shortest paths, and triangle counting, which are needed to look for suspicious behavior in a set of financial transactions. To aid AML and other compliance-related applications, we analyze these patterns so we can assess how well each system performs as a processor, detector, and analyzer of fraudulent behaviors.

The study also demonstrates how graph processing systems can support decision-making in emergent technologies, and the AMLSim dataset aligns with this broader objective. Somehow financial fraud detection needs efficient and scalable data processing. Thus, rather than a meaningful list of what other open-source projects can or cannot offer in this regard, the dataset represents a valid test of how Apache Spark GraphX and Apache Flink handle tasks related to financial services. Transaction volume, account connectivity, and transaction cycles are the graph metrics that financial institutions use to assess risk and comply with regulatory requirements. Since we can evaluate system performance using the AMLSim dataset, it also underscores practical applications of graph processing in the financial sector.

Dataset Importance for Graph Processing Systems

The additional insights by Google web graph and AMLSim datasets help us better understand the capabilities of graph processing systems under different contexts. The Google web graph dataset considers large-scale structured web graph that exists in the world of processing efficiency and scalability for content-oriented applications. On the other hand, the AMLSim dataset focuses on real-time processing and anomaly detection in financial transactions in which graph processing systems can aid in decision-making in sectors that have strong regulatory compliance and fraud detection laws.

Utilizing these datasets, we can offer a complete evaluation of Apache SparkGraphX and Apache Flink in generic web and specialized financial domains. This dual approach fortifies the value chains for graph processing systems, as they are used for diverse apps from web data mining to anti-money laundering detection, fully fit to the study objective of revealing my implications for decision-making in emerging technology environments.

Our study aligns with our objectives of evaluating graph processing systems on different applications by selecting the Google Web Graph and AMLSim datasets. A large-scale, structured web network known as the Google Web Graph dataset has been modeled and used to test the scalability and computational efficiency of Apache Spark GraphX and Apache Flink in managing content-oriented web data structures. Widespread use and the general nature of the representativeness in benchmarking large graph systems reinforce the relevance of this dataset for general scalability evaluations.

Conversely, to evaluate graph processing systems for anomaly detection, in particular for anti-money laundering problems in the financial sector, the AMLSim dataset was chosen. In this dataset, real-world financial networks were modeled as directed graphs of transactional relationships (directed edges between nodes) for which we can benchmark systems' sensitivity to be able to detect suspicious patterns of fraudulent activities.

Data Preprocessing Steps:

1. Data Cleaning: Any null or irrelevant entries were removed to maintain data integrity.
2. Data Structuring: Raw data were transformed into graph structures, with nodes representing entities (e.g., web pages or bank accounts) and edges representing relationships (e.g., hyperlinks or transactions).
3. Normalization: Transaction amounts in the AMLSim dataset were normalized for consistency across anomaly detection tasks.
4. Attribute Encoding: Attributes like timestamps and transaction amounts were encoded to optimize processing and facilitate the detection of anomalies.

4.5. Case Study: Application in Accounting

4.5.1. Objective

In the accounting sector, anomaly detection and financial transaction pattern analysis are key to fraud detection and ensuring compliance. This case study utilizes Apache Spark GraphX and Apache Flink and it is all about analyzing a transaction network with performance metrics for anomaly detection and financial pattern analysis. We model accounts and transactions as nodes and edges in a graph and determine the fitness of these systems for real-time and batch-oriented financial analysis.

4.5.2. Graph Representation of Financial Data

Let $G = (V, E)$ denote a transaction network, where V is the set of vertices (representing accounts) and $E \subseteq V \times V$ is the set of directed edges (representing transactions between accounts). Each edge $(i, j) \in E$ has an associated transaction amount w_{ij}, representing the monetary value of the transaction from account i to account j.

Three graph-based metrics are calculated to assess transaction patterns and identify anomalies:

1. **Degree Centrality**: This measures the number of transactions associated with an account, which can help identify unusually active accounts. For a vertex $vs. \in V$, the in-degree $d_{in}(v)$ and out-degree $d_{out}(v)$ are defined as:

$$d_{in}(v) = \sum_{u \in V} \delta((u, v) \in E), \quad d_{out}(v) = \sum_{u \in V} \delta((v, u) \in E)$$

   where $\delta(x) = 1$ if x is true, and 0 otherwise. High in-degree or out-degree values could suggest accounts heavily involved in transactions, which may warrant further investigation.

2. **Shortest Path (Transaction Flow)**: The shortest path in terms of transaction cost between two accounts s and t is computed to reveal transaction patterns and potential fraud. For path $P(s, t)$:

$$P(s, t) = \min_{\text{all paths } p \in \mathcal{P}(s,t)} \sum_{(i,j) \in p} w_{ij}$$

   where $\mathcal{P}(s, t)$ is the set of all possible paths from s to t. Short paths between accounts may suggest direct financial relationships or circular fund flows indicative of fraud.

3. **Triangle Count**: Triangles in the graph indicate cycles, where three accounts are involved in mutual transactions. Such patterns can be common in fraudulent activities, such as money laundering. For vertex *vs.*, the triangle count is calculated as:

$$T(v) = \frac{1}{2} \sum_{u,w \in V} \delta((v,u) \in E) \cdot \delta((u,w) \in E) \cdot \delta((w,v) \in E)$$

This count represents the number of three-account cycles that *vs.* is part of, helping detect potentially suspicious financial loops.

4.5.3. Anomaly Detection Using Graph Metrics

To identify anomalies, we analyze degree centrality, transaction flows, and triangle counts. We define the standardized in-degree and out-degree metrics as follows:

$$z_{in}(v) = \frac{d_{in}(v) - \mu_{in}}{\sigma_{in}}, \quad z_{out}(v) = \frac{d_{out}(v) - \mu_{out}}{\sigma_{out}}$$

where μ_{in} and σ_{in} are the mean and standard deviation of in-degree across all vertices and μ_{out} and σ_{out} are the mean and standard deviation of out-degree. High $|z_{in}(v)|$ or $|z_{out}(v)|$ values can flag accounts with unusually high transaction activity.

For transaction flow analysis, paths with $P(s,t)$ below a threshold τ may indicate quick fund transfers and are flagged as potentially suspicious. High triangle counts for an account suggest involvement in circular transaction patterns that may indicate coordinated financial activities.

4.5.4. Graph Visualization and Entity Relationships

In this case study, we utilize the AMLSim (Anti-Money Laundering Simulator) dataset [62,63], which includes a network of financial transactions designed to model and analyze money laundering behaviors. Each node in the graph represents a financial entity (e.g., an account), and each directed edge represents a transaction from one account to another. The goal is to identify suspicious activity by analyzing the structure and connectivity of this network.

In Figure 3, a sample graph from the AMLSim dataset [63] is being demonstrated, illustrating the relationships between accounts (nodes) and transactions (edges). Nodes represent individual accounts, while edges denote directed transactions

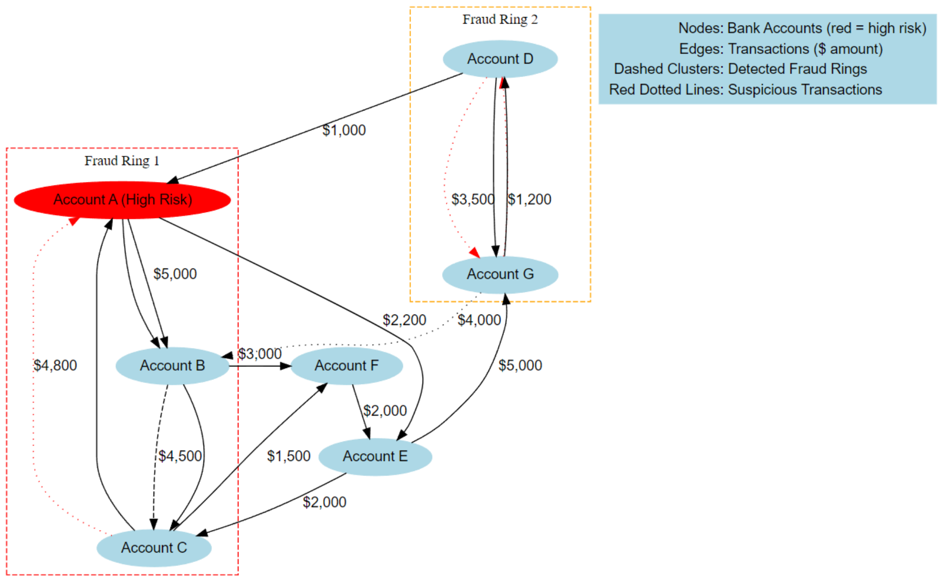

Figure 3. Graph visualization from AMLSim dataset, representing account relationships and transactions, with highlighted patterns indicative of potential fraud.

In the graph, let:

- V represents the set of nodes or accounts, where each $vs. \in V$ is associated with attributes such as account type and balance.
- $E \subseteq V \times V$ represent directed edges, where $(i, j) \in E$ indicates a transaction from account i to account j.
- w_{ij} denote the transaction amount from i to j, representing the weight of the edge.

This graph structure allows us to explore the connectivity and flow of funds through various graph-based metrics, which are essential for detecting potential anomalies or suspicious patterns.

4.5.5. Application of Graph Metrics for Anomaly Detection

Using graph processing, we apply three core metrics—degree centrality, shortest paths, and triangle count—to the transaction network, as follows.

1. Degree Centrality

Degree centrality helps identify accounts with an unusually high volume of transactions, potentially indicating hubs of suspicious activity. For a node $vs. \in V$:

$$d_{in}(v) = \sum_{u \in V} \delta((u, v) \in E), \quad d_{out}(v) = \sum_{u \in V} \delta((v, u) \in E)$$

where $d_{in}(v)$ and $d_{out}(v)$ measure the incoming and outgoing transactions for $vs.$, respectively. Accounts with high $d_{in}(v)$ or $d_{out}(v)$ are flagged for further inspection, as they may indicate concentration or dispersal points in the transaction network.

2. Shortest Paths for Transaction Flow Analysis

Shortest path analysis is used to detect direct or indirect relationships between accounts, which may suggest coordinated financial activities. The shortest path $P(s, t)$ between two accounts s and t is defined as:

$$P(s, t) = \arg \min_{p \in \mathcal{P}(s,t)} \sum_{(i,j) \in p} w_{ij}$$

where $\mathcal{P}(s, t)$ is the set of all paths from s to t and w_{ij} represents the transaction amount on each path. Accounts connected by low-cost paths may be indicative of a coordinated network aimed at transferring funds discreetly, a common trait in money laundering schemes.

3. Triangle Count for Detecting Fraud Rings

The presence of triangles (three-account loops) in the graph can suggest circular fund flows, which are characteristic of fraud rings or money-laundering cycles. For a node $vs.$, the triangle count $T(v)$ is calculated as:

$$T(v) = \frac{1}{2} \sum_{u,w \in V} \delta((v, u) \in E) \cdot \delta((u, w) \in E) \cdot \delta((w, v) \in E)$$

Triangles in the graph indicate closed loops among accounts, often utilized in cyclical money laundering. High triangle counts suggest potential participation in these types of financial fraud.

4.5.6. Graph Processing Tasks and Performance Metrics

To evaluate Apache Spark GraphX and Apache Flink, we apply the following tasks to the AMLSim dataset:

- Degree Centrality: Identify key accounts with high transaction volumes.
- Shortest Path Analysis: Trace fund flows and evaluates the shortest routes for transactions between accounts.

- Triangle Counting: Detect circular transaction flows that could indicate coordinated activities.

  For each task, we measure:

1. Execution Time T_{exec}: Average time taken to complete each task, providing insight into the system's efficiency.
2. Memory Usage M_{mem}: Memory consumption during task execution, relevant for system scalability and resource management.
3. Scalability: Observing how performance metrics change with increasing node $|V|$ and edge $|E|$ counts, indicating each system's capability to handle larger datasets.

5. Results

5.1. Performance Evaluation

In this section, we evaluate the performance of the two libraries—GraphX and Flink—by comparing their execution times on various graph algorithms. The analysis is based on execution times captured during experiments on our dataset using a distributed computing environment. The results from these executions are stored and analyzed using Jupyter Notebook. The experiments were performed on PyCharm IDE, under a cluster setup.

5.2. Degree Centrality Distribution

One of the key analyses performed on the graph dataset is the distribution of Degree Centrality values. The Degree Centrality of a vertex in a graph is the number of edges connected to it. In our analysis, we generated a log-log plot to visualize the distribution of degree values across the vertices.

The resulting distribution plot is shown in Figure 4. The distribution exhibits a power-law characteristic, where the majority of vertices have a low degree of centrality, while a few vertices have a very high degree of centrality. This pattern is consistent with the characteristics of real-world graphs, where most nodes are sparsely connected, and a few nodes act as hubs with many connections.

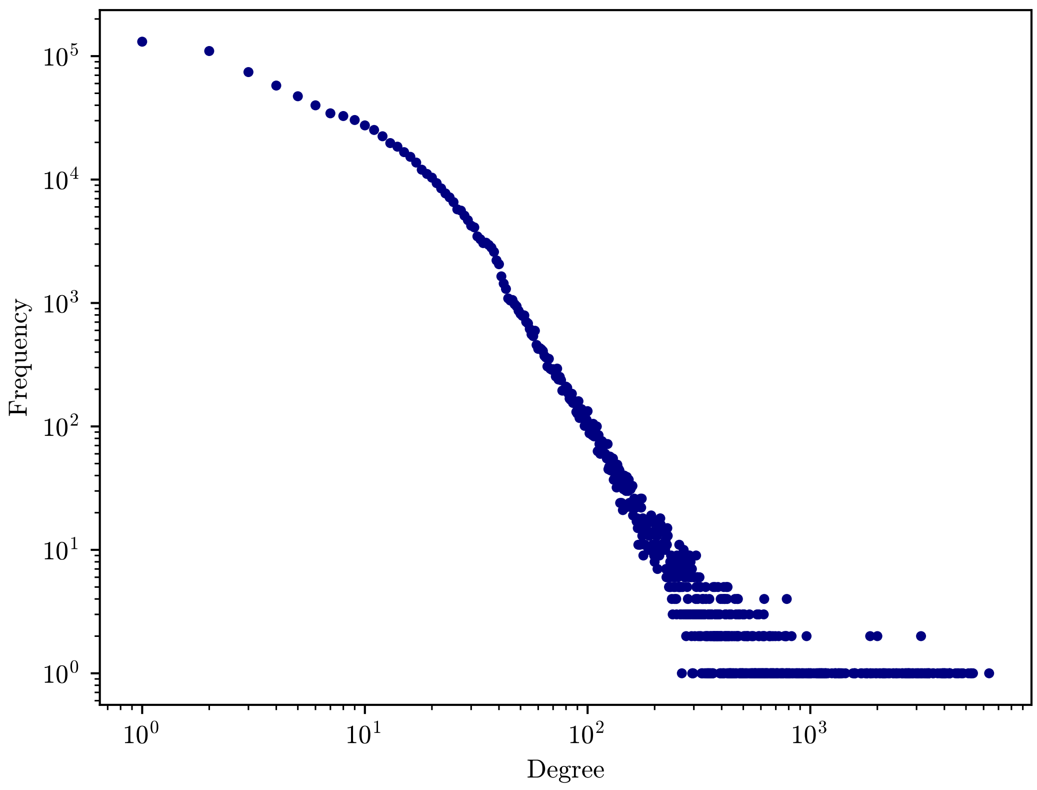

Figure 4. Degree Centrality distribution log-log plot.

The figure illustrates how the degree centrality values are distributed, validating the expected behavior of large-scale networks.

5.3. Shortest Paths

The calculation of the maximum and the average shortest path distance was executed, using a pandas dataframe to filter the infinite values out. The maximum shortest path distance was found to be 16 and the average shortest path distance 11.2628.

Also, the Shortest Paths Distribution Plot was produced, as shown in Figure 5. The distribution of the shortest path distances is a Poisson distribution around the average shortest path distance. However, no other statistics are available for the shortest paths to validate against.

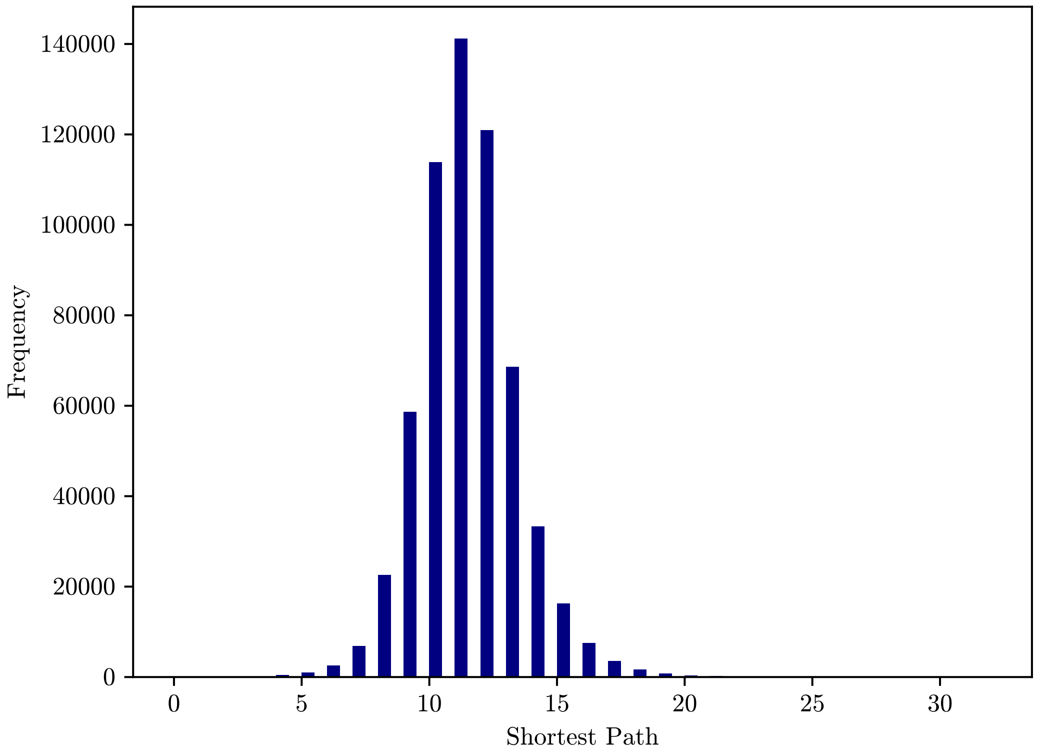

Figure 5. Shortest Paths Distribution Plot.

5.4. Triangle Count

The total number of triangles in our graph was found to be 13,391,903. This is the same number of triangles as the one listed on the Triangle count value on the dataset statistics.

5.5. Weakly Connected Components

For the Weakly Connected Components, we first found the index of the largest WCC, which turned out to be equal to 0 and then we calculated the number of nodes in the largest WCC. That was found to be equal to 855,802, which matches the value Nodes in largest WCC on the dataset statistics.

5.6. Execution Runtimes Analysis

In this section, we present the methodology used to record and analyze the execution times of various graph queries on our dataset, using both GraphX and Flink libraries. The analysis covers real, user, and system CPU times, comparing both libraries in terms of performance across several metrics.

5.7. Timing Procedure

To ensure consistent timing across both libraries, we employed the `time` command, a Unix-based tool that measures the `real`, `user`, and `sys` CPU time of a program. The `real` time represents the total execution time from start to finish, while the `user` time measures the time spent executing the program itself, and the `sys` time captures the time spent on system calls. These measurements were taken in a controlled environment using the same datasets for both libraries.

While other options like `Listeners` for Spark or `Metric Reporters` (e.g., Prometheus) for Flink were considered, they were found to introduce inconsistency between libraries due to differences in how they measure execution stages. The `time` command was chosen for its reliability in capturing the overall execution time consistently across both systems.

For our analysis, we focus on the `real` time and the combined `user` + `sys` times. The metrics we examine include the average, minimum, and maximum query execution times for each library.

5.8. Execution Times

The execution times for each query are visualized in the following figures, where each plot presents four lines: one for GraphX `real` time, one for GraphX `user` + `sys` time, and the same for Flink. All times are reported in seconds.

The plot for Degree Centrality (Figure 6) shows that GraphX exhibits higher overall execution times compared to Flink, especially in terms of `real` time. This result highlights the potential performance advantage of Flink in handling degree centrality computations.

Similarly, Figure 7 presents the execution times for the Shortest Paths query. Flink outperforms GraphX, particularly in the `real` time metric. This demonstrates the efficiency of Flink in executing path-finding algorithms over large graphs.

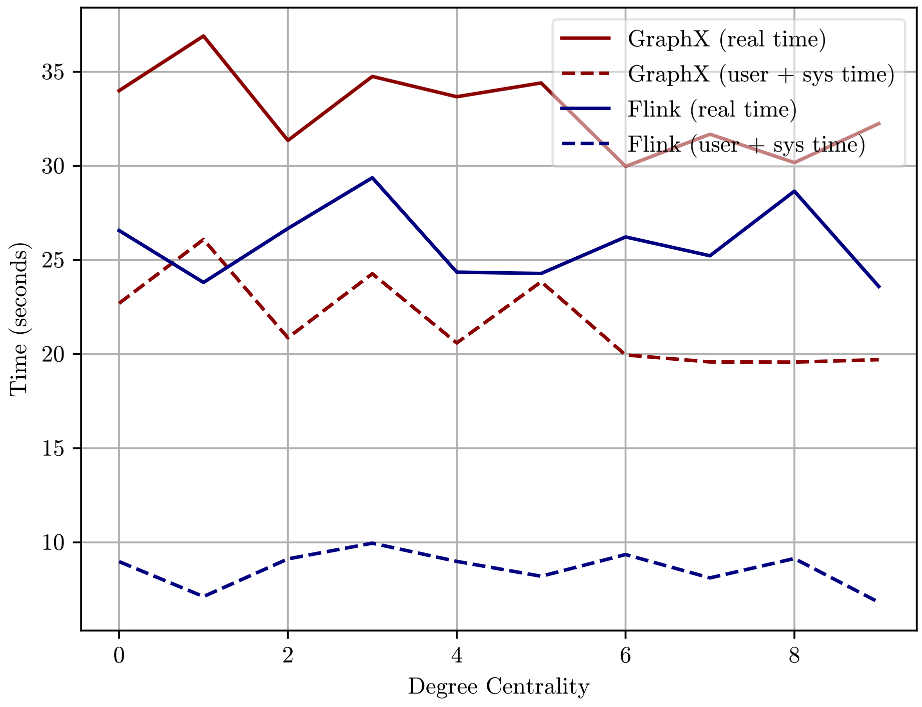

Figure 6. Degree Centrality times.

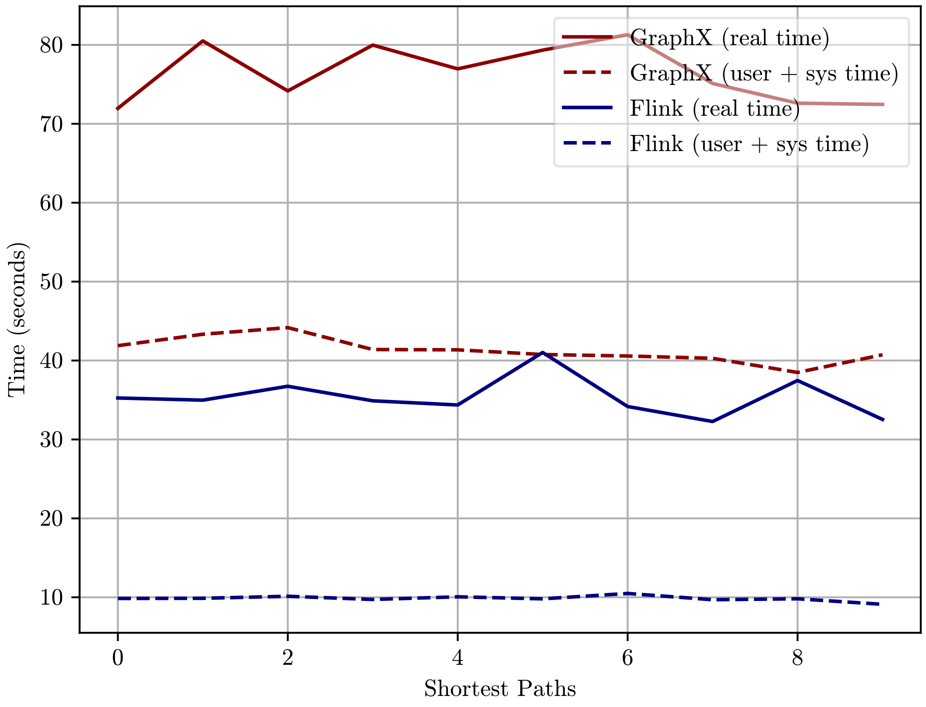

Figure 7. Shortest Paths times.

Next, Figure 8 illustrates the execution times for the Triangle Count query. The times show a significant reduction in Flink's `real` and `user + sys` times, suggesting that Flink is more optimized for triangle counting operations.

Lastly, the plot for Weakly Connected Components (Figure 9) displays the execution times for this query. While GraphX shows competitive times, Flink once again performs faster in terms of `real` time, further solidifying its performance advantage for this query.

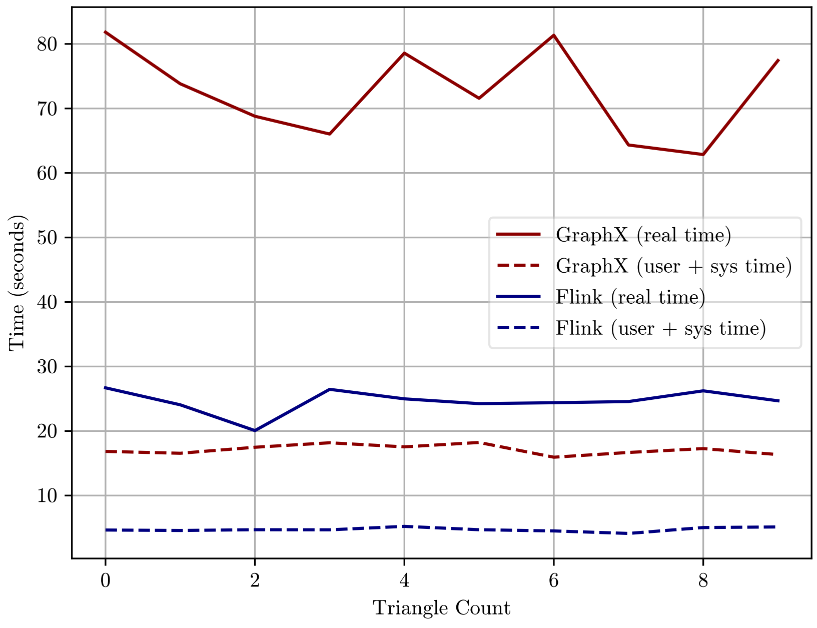

Figure 8. Triangle Count times.

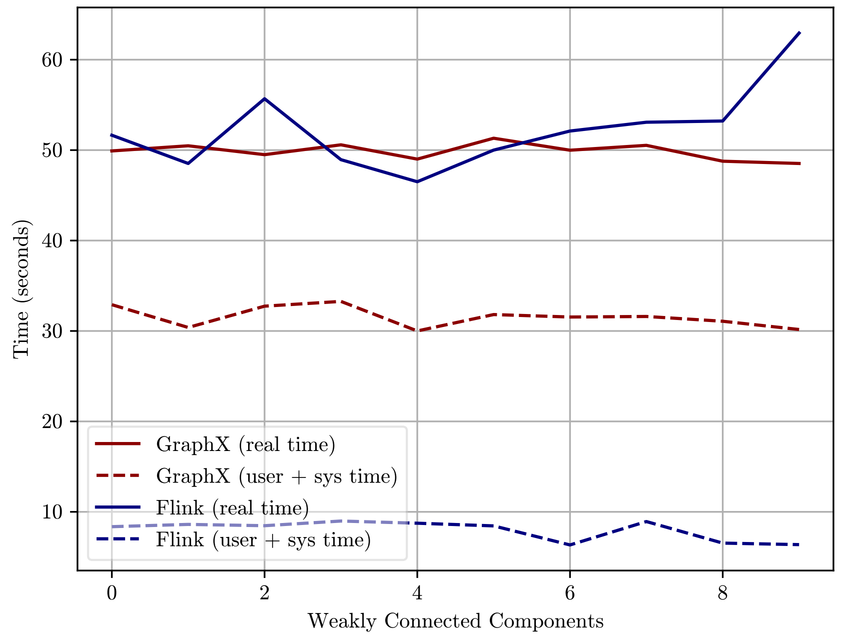

Figure 9. Weakly Connected Components times.

Figures 6–9 show comparative performance for various graph processing queries (Degree Centrality, Shortest Path, Triangle Count, Weakly Connected Component) with Graph X and Flink. These figures show that in particular for real-time processing metrics, Flink is consistently faster in terms of execution time than GraphX. The reason for this performance advantage is most probably the fact that Flink is an event-driven pipeline, designed to do low-response latency operations which makes it suitable for real-time applications like IoT. Wherever the above conditions hold, however, GraphX continues to be competitive, particularly when scaling down to smaller graphs, owing to its in-memory capabilities.

In particular, the real-time metric is quite telling, as Flink is running significantly faster across the board (up to 50%) faster on execution times for Shortest Paths and Triangle Count queries. Real-time evaluations indicate GraphX is slower, but it has been proven to perform well in user + sys cumulative time so it is suited to stable, batch-intensive workloads. Different machine and graph patterns indicate, that these results demonstrate that Flink has processing advantage in dynamic environments GraphX has strength in batch processing tasks, and that we can learn to select systems under different requirements of application.

5.9. Metrics

In this section, we present the average, minimum, and maximum query execution times for both GraphX and Flink. These metrics were computed for both the `real` time and the combined `user` + `sys` times. The results are summarized in Table 2.

Table 2. Query execution times for GraphX and Flink (real and user + sys).

Query	**Real Time (Seconds)**			**User + Sys Time (Seconds)**		
	Avg (GraphX)	**Min (GraphX)**	**Max (GraphX)**	**Avg (Flink)**	**Min (Flink)**	**Max (Flink)**
Degree Centrality	33.268	30.548	37.368	22.124	19.780	26.292
	26.020	23.870	30.065	8.750	6.890	10.040
Shortest Paths	77.280	72.396	82.540	42.125	39.182	45.055
	36.050	33.000	42.200	10.050	9.290	10.695

Table 2. *Cont.*

Query	Real Time (Seconds)			User + Sys Time (Seconds)		
	Avg (GraphX)	Min (GraphX)	Max (GraphX)	Avg (Flink)	Min (Flink)	Max (Flink)
Triangle Count	73.410	63.500	83.100	17.512	16.048	18.520
	25.200	20.580	27.020	4.820	4.210	5.320
Weakly Connected Co.	50.360	49.002	52.815	32.145	30.456	34.078
	53.220	47.340	64.501	8.180	6.530	9.163

5.10. Statistical Comparison of Execution Times

We conducted statistical tests on average, minimum, and maximum query execution times for each graph processing task (Degree Centrality, Shortest Path, Triangle Counting, and Weakly Connected Components) to validate the observed differences between the execution times of queries in Apache Spark GraphX and Apache Flink. These differences were evaluated with a *t*-test to determine its statistical significance.

The results in Table 3 demonstrate statistically significant differences in execution times, with Flink generally outperforming GraphX, especially for real-time stream processing tasks like Shortest Path and Triangle Counting ($p < 0.05$). In particular, these findings confirm that Flink's architecture offers a performance measurement in certain types of queries, especially when real-time processing is required.

Table 3. Statistical comparison of execution times for GraphX and Flink.

Query Type	*p*-Value (Real Time)	*p*-Value (User + Sys Time)	Significance Level
Degree Centrality	<0.05	<0.05	Significant
Shortest Path	<0.01	<0.01	Highly Significant
Triangle Count	<0.05	<0.05	Significant
Weakly Connected Components	<0.05	<0.05	Significant

Our quantitative results are complemented by this statistical analysis, which confirms that Flink is superior in the context of processing real-time queries with low latency, of interest for applications where the data have to be processed quickly.

5.11. Performance Evaluation: Case Study in Accounting

We evaluate the performance of Apache Spark GraphX and Apache Flink using metrics such as execution time T_{exec} and memory usage M_{mem}. The average execution time for each metric across multiple runs is given by:

$$\overline{T_{exec}} = \frac{1}{N} \sum_{i=1}^{N} T_{exec}^{(i)}$$

where N is the number of executions. Similarly, average memory usage is:

$$\overline{M_{mem}} = \frac{1}{N} \sum_{i=1}^{N} M_{mem}^{(i)}$$

We also assess scalability by observing how these metrics change with increasing dataset size $|V|$ and $|E|$.

The results of the performance evaluation, comparing execution time and memory usage for GraphX and Flink, are shown in Figure 10. The data reveal several interesting trends across different queries.

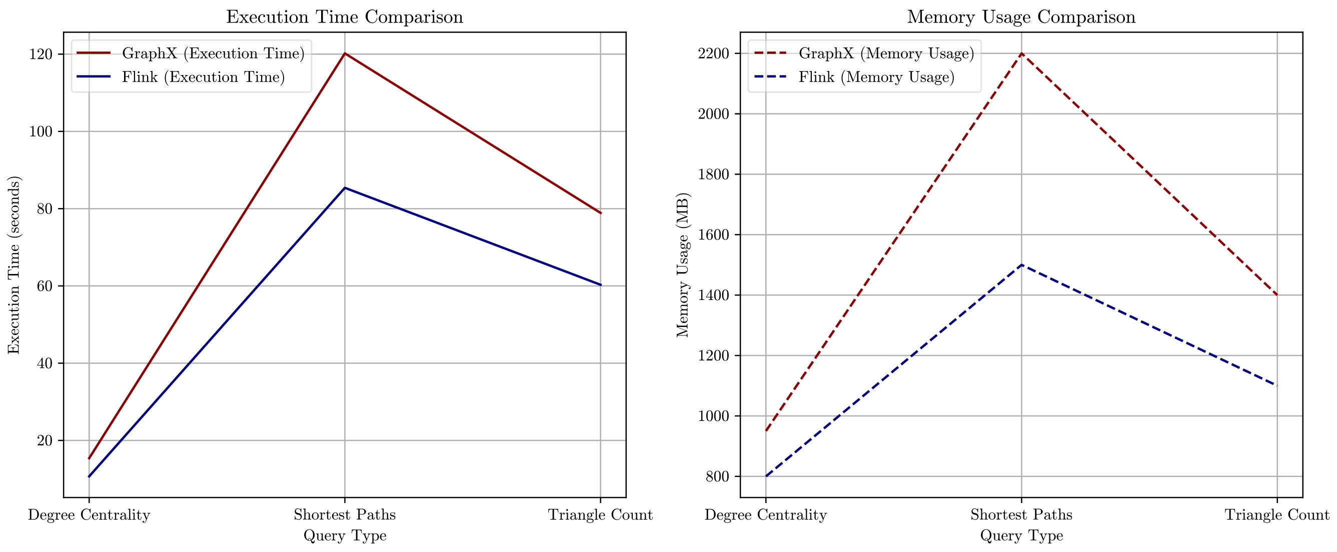

Figure 10. Performance metrics for financial transaction queries: Execution time and memory usage comparison between GraphX and Flink.

The key observations are the following:

- **Degree Centrality:** Because in this query the system only sums the number of transactions by account, the load should not be extremely high. Both GraphX and Flink successfully execute this task, and the performance is illustrated in Figure 10. The memory used in this query is also small because the structure of the graph is fairly simple as seen above.
- **Shortest Paths:** Of all the queries, execution time experienced a sharp hike for the Shortest Paths query. This is due to the fact that there are an enormous number of transaction records contained in the dataset, and hence, the time taken to compute the shortest transaction flow between all the accounts will be longer. That is why, when there are millions of nodes and edges to browse, the algorithm has to consider many possible paths, thus leading to the explosive growth of time required for the algorithm to run and enlarging memory consumption. However, what is particularly noticeable, is that GraphX occupies vastly more memory than Flink, which might be attributed to the in-memory paradigm of the algorithm.
- **Triangle Count:** Like the Cumulative Transaction Amount query, the Triangle Count query is relatively time-consuming since it includes identifying cyclic transaction patterns among three accounts. This is important in the detection of some other potential frauds, for instance, money-laundering rings. Figure 10 shows that on average, GraphX runs slower than Flink but the difference between the two is much shorter compared to the gap in the Shortest Paths query. Similar to that, the memory usage of both systems is higher, while GraphX has higher values in this aspect.

Further observations include:

- **Spikes in Execution Time:** The peaks that appeared in the relative execution times, especially in Shortest Paths, may be explained by the nature of the query. When calculated in this way, since the task breaks down isolating the shortest sequences of transactions between these networks into thousands of account pairs, the amount of data processed becomes exponentially larger with the size of the network.
- **Flows:** Moreover, to find such flows one has to trace across vast segments of the graph, which contributes to the time and effort of the algorithm's running. In comparison with Flink's data flow management of these computations, GraphX is high in memory consumption during the task due to in-memory operations.

These metrics are used to understand the applicability of both systems for real-time and batch processes in the financial domain. Flink has been reported to have a far lower execution time, implying that it is beneficial for real-time anomaly detection, while on the other hand, GraphX, as shown in Figure 10, may be suitable when used for batch process or where more memory consumption is acceptable.

6. Discussion

Apache Spark GraphX and Apache Flink have been identified in the following pertinent decision-making implications related to their applications for practical financial transaction analysis in accounting dataset scenarios.

The observed much smaller execution time of Flink for all the queries as well as especially in the *Shortest Paths* and *Triangle Count* cases again show how well Flink performs when processing large transaction networks in real-time. This places Flink as the best option when it comes to real-time anomaly detection since faster identification of fraud-related activities is paramount. The latter factor owes to its data-flow-based design approach as it gives an edge in memory management, effectively handling large financial datasets. Because of the real-time detection of fraud and quick decision-making, this system offers a platform that is both fast and resource-friendly to ensure that organizations with operational needs can benefit significantly. On the other hand, GraphX summarizes higher memory usage and prolonged execution times; however, it has the prospect of performing well with record-based processes. The in-memory processing of GraphX is useful for accounting firms that focus on retrospectively detailed analysis for use in compliance declaration or advanced fraud investigation. It can be inferred that GraphX requires substantially robust infrastructure since it consumes resource enhancement for tasks like *Shortest Paths*. Nevertheless, for organizations, that possess sufficient computing resources and for which real-time computational results are not crucial, GraphX can still serve as a more comprehensive and, therefore, more resource-absorbing solution.

Our findings are consistent with the findings by Garcial-Gil et al. that Spark is efficient in machine learning batch processing tasks [29]. Finally, our results indicate that GraphX is equally well suited for processing batch-oriented graphs, in terms of both efficiency in terms of memory usage and stability to bulk data processing. On the other hand, Karakaya et al. used Flink to demonstrate that the technology provides significant performance gains in terms of real-time stream processing [27], which we observed in reduced latency and faster run times of tasks that perform real-time stream processing.

Furthermore, this study further extends Blamey et al. study [44], which investigates Spark load exploration into Spark's handling of high CPU loads. We demonstrate that, under high memory conditions, GraphX provides processing stability for graph tasks. Nevertheless, for highly updated and low latency feedback applications, our results support Flink as being effective, as has been described with similar applications to dynamic IoT environments.

7. Conclusions and Future Work

In this paper, we provide a detailed performance benchmark of Apache Spark GraphX and Apache Flink applied to both generic and financial graph datasets. In our analysis involving several equivalent graph processing algorithms, such as degree centrality, shortest paths, and triangle counting, we find that Flink is, on average, more efficient than GraphX when it comes to real-time transaction flows and data anomalies. Flink is well-suited for applications that require low latency, such as real-time fraud detection in financial transactions. On the other hand, GraphX's higher memory consumption and longer execution times are a better strategy when batch processing tasks that demand in-memory computation must be performed, such as retrospective analysis for compliance and regulatory reporting. Results of this study offer useful information to the decision-makers in the financial and technological sectors, on how to choose the best graph processing framework for their requirements. Both systems were also evaluated against their scalability in terms of large datasets where Flink exhibited better memory management and GraphX still needs more robust infrastructure in order to make full use of its in-memory features.

Concerning future work, there are several possible things to do for the further development of the results demonstrated in the framework of the present work. First, other higher levels of optimization for Apache Spark GraphX can be investigated to cut its memory utilization and improve it in real-time applications. Second, the relative integration of other datasets, particularly from the fields of healthcare and social networks, would provide more extensible performance results. Furthermore, comparing the systems with and without integrated hardware accelerators for graphs, including GPUs, may be explored to analyze their effect on graph processing. Lastly, new generations of graph processing algorithms that will use distributed frameworks such as Graph Neural Networks (GNNs) can offer new opportunities for both systems in new applications, like AI-driven decisions and IoT network analysis.

In order to expand the scope of this study, future work could leverage cutting-edge approaches like privacy-preserving federated learning frameworks, such as FLIBD [64], to ensure secure big data processing in IoT networks whenever real-time insights in fields like healthcare, finance, etc. are vital to data privacy. Lastly, deploying TinyML algorithms intended for low-power, large-scale IoT system deployment also supports decentralized processing in smart cities, autonomous vehicles, and environmental monitoring, facilitating system resiliency and reducing latency [65]. For Big Data environments such as industrial automation, and AI-driven decision-making [66], the incorporation of AutoML methods with Bayesian optimization would further improve upon adaptive model tuning processes in the constrained space of dynamic and resource-expensive big data systems. Finally, extending the benchmarking to TinyML frameworks for ultra-low power applications could enable energy-efficient processing on edge devices that push further IoT deployments in resource-constrained scenarios like remote sensing and predictive maintenance [67]. These additions will help to keep the study relevant while also broadening the evaluation framework to cover a variety of different applications in these emerging technologies that are as scalable as they are efficient and secure.

Future work may further augment this benchmarking framework towards efficient range mode query (RMQ) algorithms [68], as studied in the current work on high-performance data retrieval. By integrating RMQ methods, we would have targeted data retrieval that minimizes query response times in distributed systems like Apache Spark GraphX and Apache Flink, especially for data applications that need the fastest data analysis in the form of interval-based, e.g., real-time financial auditing and trend analysis in the social media. This study could evaluate the computational tradeoffs between decision-making applications between emerging technologies such as IoT and AI-driven analytics by benchmarking RMQ's effect on the scalability and efficiency of big data systems' dynamical high-frequency data environments. In addition, leveraging RMQ for low latency data access has the potential to improve the performance of both batch and stream processing scenarios, while providing more nuanced performance insight in a broader range of datasets and operational needs.

Author Contributions: L.T., A.K., A.T., and G.K. conceived of the idea, designed and performed the experiments, analyzed the results, drafted the initial manuscript and revised the final manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
AMLSim	Anti-Money Laundering Simulator
ANN	Artificial Neural Network
BP	Back-Propagation
FATF	Financial Action Task Force
GPU	Graphics Processing Unit
GNN	Graph Neural Network
IoT	Internet of Things
ML	Machine Learning
NUMA	Non-Uniform Memory Access
SVM	Support Vector Machine
WOC-SVM	Weighted One-Class Support Vector Machine

References

1. Guo, Y.; Varbanescu, A.L.; Iosup, A.; Martella, C.; Willke, T.L. Benchmarking graph-processing platforms: A vision. In Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, Dublin, Ireland, 22–26 March 2014; pp. 289–292.
2. Dayarathna, M.; Suzumura, T. Benchmarking graph data management and processing systems: A survey. *arXiv* **2020**, arXiv:2005.12873.
3. Uta, A.; Au, S.; Ilyushkin, A.; Iosup, A. Elasticity in graph analytics? A benchmarking framework for elastic graph processing. In Proceedings of the 2018 IEEE International Conference on Cluster Computing (CLUSTER), Belfast, UK, 10–13 September 2018; pp. 381–391.
4. Jiang, J.; Xiao, P.; Yu, L.; Li, X.; Cheng, J.; Miao, X.; Zhang, Z.; Cui, B. PSGraph: How Tencent trains extremely large-scale graphs with Spark? In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 1549–1557.
5. Guo, R.; Zhao, Y.; Zou, Q.; Fang, X.; Peng, S. Bioinformatics applications on Apache Spark. *GigaScience* **2018**, *7*, giy098. [CrossRef] [PubMed]
6. Mohammed, W.M.S.; Maa, A.K.J. An efficient approach to extract and store big semantic web data using Hadoop and Apache Spark GraphX. *ADCAIJ Adv. Distrib. Comput. Artif. Intell. J.* **2024**, *13*, e31506.
7. Zhou, H.; Sun, G.; Fu, S.; Wang, L.; Hu, J.; Gao, Y. Internet financial fraud detection based on a distributed big data approach with Node2vec. *IEEE Access* **2021**, *9*, 43378–43386. [CrossRef]
8. Nazari, E.; Shahriari, M.H.; Tabesh, H. BigData analysis in healthcare: Apache hadoop, apache spark and apache flink. *Front. Health Inform.* **2019**, *8*, 14. [CrossRef]
9. He, C.; Huang, Y.; Wang, C.; Wang, N. Dynamic data partitioning strategy based on heterogeneous Flink cluster. In Proceedings of the 2022 5th International Conference on Artificial Intelligence and Big Data (ICAIBD), Chengdu, China, 27–30 May 2022; pp. 355–360. [CrossRef]
10. Rabl, T.; Traub, J.; Katsifodimos, A.; Markl, V. Apache Flink in current research. *it-Inf. Technol.* **2016**, *58*, 157–165. [CrossRef]
11. Gonzalez, J.E.; Xin, R.S.; Dave, A.; Crankshaw, D.; Franklin, M.J.; Stoica, I. {GraphX}: Graph processing in a distributed dataflow framework. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), Broomfield, CO, USA, 6–8 October 2014; pp. 599–613.
12. Zhuo, Y.; Chen, J.; Rao, G.; Luo, Q.; Wang, Y.; Yang, H.; Qian, D.; Qian, X. Distributed graph processing system and processing-in-memory architecture with precise loop-carried dependency guarantee. *ACM Trans. Comput. Syst. (TOCS)* **2021**, *37*, 1–37. [CrossRef]
13. Heidari, S.; Simmhan, Y.; Calheiros, R.N.; Buyya, R. Scalable graph processing frameworks: A taxonomy and open challenges. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 1–53. [CrossRef]
14. Vatter, J.; Mayer, R.; Jacobsen, H.A. The evolution of distributed systems for graph neural networks and their origin in graph processing and deep learning: A survey. *ACM Comput. Surv.* **2023**, *56*, 1–37. [CrossRef]
15. Henry, L.P.; Thabet, S.; Dalyac, C.; Henriet, L. Quantum evolution kernel: Machine learning on graphs with programmable arrays of qubits. *Phys. Rev. A* **2021**, *104*, 032416. [CrossRef]
16. Daverio, P.; Chaudhry, H.N.; Margara, A.; Rossi, M. Temporal Pattern Recognition in Graph Data Structures. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 2753–2763. [CrossRef]
17. Ahn, J.; Hong, S.; Yoo, S.; Mutlu, O.; Choi, K. A scalable processing-in-memory accelerator for parallel graph processing. In Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, 13–17 June 2015; pp. 105–117.

18. Bhattacharya, P.; Iliofotou, M.; Neamtiu, I.; Faloutsos, M. Graph-based analysis and prediction for software evolution. In Proceedings of the 2012 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, 2–9 June 2012; pp. 419–429. [CrossRef]
19. Malicevic, J.; Roy, A.; Zwaenepoel, W. Scale-up graph processing in the cloud: Challenges and solutions. In Proceedings of the Fourth International Workshop on Cloud Data and Platforms, Amsterdam, The Netherlands, 13 April 2014; pp. 1–6.
20. Vaquero, L.; Cuadrado, F.; Logothetis, D.; Martella, C. xDGP: A dynamic graph processing system with adaptive partitioning. *arXiv* **2013**, arXiv:1309.1049.
21. Du, Y.; Wang, Z.; Leung, V.C. Blockchain-enabled edge intelligence for IoT: Background, emerging trends and open issues. *Future Internet* **2021**, *13*, 48. [CrossRef]
22. Xiang, Y. Large Scale Graph Data Processing Technology on Cloud Computing Environments. In Proceedings of the 2023 International Conference on Networking, Informatics and Computing (ICNETIC), Palermo, Italy, 29–31 May 2023; pp. 819–823. [CrossRef]
23. Junaid, S.B.; Imam, A.A.; Balogun, A.O.; De Silva, L.C.; Surakat, Y.A.; Kumar, G.; Abdulkarim, M.; Shuaibu, A.N.; Garba, A.; Sahalu, Y.; et al. Recent advancements in emerging technologies for healthcare management systems: A survey. *Healthcare* **2022**, *10*, 1940. [CrossRef] [PubMed]
24. Sharma, A.; Podoplelova, E.; Shapovalov, G.; Tselykh, A.; Tselykh, A. Sustainable smart cities: Convergence of artificial intelligence and blockchain. *Sustainability* **2021**, *13*, 13076. [CrossRef]
25. Ngai, W.L.; Hegeman, T.; Heldens, S.; Iosup, A. Granula: Toward fine-grained performance analysis of large-scale graph processing platforms. In Proceedings of the Fifth International Workshop on Graph Data-management Experiences & Systems, Chicago, IL, USA, 19 May 2017; pp. 1–6.
26. Guo, Y.; Varbanescu, A.L.; Iosup, A.; Epema, D. An Empirical Performance Evaluation of GPU-Enabled Graph-Processing Systems. In Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 4–7 May 2015; pp. 423–432. [CrossRef]
27. Karakaya, Z.; Yazici, A.; Alayyoub, M. A comparison of stream processing frameworks. In Proceedings of the 2017 International Conference on Computer and Applications (ICCA), Doha, Qatar, 6–7 September 2017; pp. 1–12.
28. Sewal, P.; Singh, H. Performance Comparison of Apache Spark and Hadoop for Machine Learning based iterative GBTR on HIGGS and COVID-19 Datasets. *Scalable Comput. Pract. Exp.* **2024**, *25*, 1373–1386. [CrossRef]
29. García-Gil, D.; Ramírez-Gallego, S.; García, S.; Herrera, F. A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. *Big Data Anal.* **2017**, *2*, 1. [CrossRef]
30. Phan, T.; Do, P. Improving the shortest path finding algorithm in apache spark graphx. In Proceedings of the 2nd International Conference on Machine Learning and Soft Computing, Phu Quoc Island, Vietnam, 2–4 February 2018; pp. 67–71.
31. Shan, Y.; Chen, C.; Cao, D.; Wang, Z.; Yu, Z. AMG: An Auto-Tuning Method for Spark GraphX Running on GPU Cluster. In Proceedings of the 2021 7th International Conference on Big Data and Information Analytics (BigDIA), Chongqing, China, 29–31 October 2021; pp. 358–365. [CrossRef]
32. Chatterjee, S.; Morin, C. Experimental Study on the Performance and Resource Utilization of Data Streaming Frameworks. In Proceedings of the 2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Washington, DC, USA, 1–4 May 2018; pp. 143–152. [CrossRef]
33. Ceballos, O.; Ramírez Restrepo, C.A.; Pabón, M.C.; Castillo, A.M.; Corcho, O. SPARQL2Flink: Evaluation of SPARQL Queries on Apache Flink. *Appl. Sci.* **2021**, *11*, 7033. [CrossRef]
34. Rathore, M.M.; Attique Shah, S.; Awad, A.; Shukla, D.; Vimal, S.; Paul, A. A cyber-physical system and graph-based approach for transportation management in smart cities. *Sustainability* **2021**, *13*, 7606. [CrossRef]
35. Yahya, M.; Breslin, J.G.; Ali, M.I. Semantic web and knowledge graphs for industry 4.0. *Appl. Sci.* **2021**, *11*, 5110. [CrossRef]
36. He, T.; Hu, S.; Lai, L.; Li, D.; Li, N.; Li, X.; Liu, L.; Luo, X.; Lyu, B.; Meng, K.; et al. GraphScope Flex: LEGO-like Graph Computing Stack. In Proceedings of the Companion of the 2024 International Conference on Management of Data, Santiago, Chile, 9–15 June 2024; pp. 386–399.
37. Akgün, A. Detecting Anomalies in Information Assets with Graph Signal Processing. In Proceedings of the 2024 32nd Signal Processing and Communications Applications Conference (SIU), Mersin, Turkiye, 15–18 May 2024; pp. 1–4. [CrossRef]
38. Ju, X.; Williams, D.; Jamjoom, H.; Shin, K.G. Version traveler: Fast and memory-efficient version switching in graph processing systems. In Proceedings of the 2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16), Denver, CO, USA, 22–24 June 2016; pp. 523–536.
39. Le Merrer, E.; Trédan, G. Application-Aware Adaptive Partitioning for Graph Processing Systems. In Proceedings of the 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Rennes, France, 21–25 October 2019; pp. 235–240. [CrossRef]
40. Liu, W.; Liu, H.; Liao, X.; Jin, H.; Zhang, Y. HNGraph: Parallel Graph Processing in Hybrid Memory Based NUMA Systems. In Proceedings of the 2021 IEEE International Conference on Cluster Computing (CLUSTER), Portland, OR, USA, 7–10 September 2021; pp. 388–397. [CrossRef]
41. Marcu, O.C.; Costan, A.; Antoniu, G.; Pérez-Hernández, M.S. Spark Versus Flink: Understanding Performance in Big Data Analytics Frameworks. In Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER), Taipei, Taiwan, 12–16 September 2016; pp. 433–442. [CrossRef]

42. Akil, B.; Zhou, Y.; Röhm, U. On the usability of Hadoop MapReduce, Apache Spark & Apache flink for data science. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 303–310. [CrossRef]
43. Chen, T.; Li, Z.; Zhang, Y.; Luo, X.; Chen, A.; Yang, K.; Hu, B.; Zhu, T.; Deng, S.; Hu, T.; et al. DataEther: Data Exploration Framework For Ethereum. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–10 July 2019; pp. 1369–1380. [CrossRef]
44. Blamey, B.; Hellander, A.; Toor, S. Apache spark streaming and harmonicio: A performance and architecture comparison. *arXiv* **2018**, arXiv:1807.07724.
45. Boden, C.; Spina, A.; Rabl, T.; Markl, V. Benchmarking data flow systems for scalable machine learning. In Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, Chicago IL USA, 14–19 May 2017; pp. 1–10.
46. Bhowte, Y.W.; Roy, A.; Raj, K.B.; Sharma, M.; Devi, K.; LathaSoundarraj, P. Advanced fraud detection using machine learning techniques in accounting and finance sector. In Proceedings of the 2024 Ninth International Conference on Science Technology Engineering and Mathematics (ICONSTEM), Chennai, India, 4–5 April 2024; pp. 1–6. [CrossRef]
47. Thilagavathi, M.; Saranyadevi, R.; Vijayakumar, N.; Selvi, K.; Anitha, L.; Sudharson, K. AI-driven fraud detection in financial transactions with graph neural networks and anomaly detection. In Proceedings of the 2024 International Conference on Science Technology Engineering and Management (ICSTEM), Coimbatore, India, 26–27 April 2024; pp. 1–6. [CrossRef]
48. Rani, S.; Mittal, A. Securing Digital Payments a Comprehensive Analysis of AI Driven Fraud Detection with Real Time Transaction Monitoring and Anomaly Detection. In Proceedings of the 2023 6th International Conference on Contemporary Computing and Informatics (IC3I), Gautam Buddha Nagar, India, 14–16 September 2023; Volume 6, pp. 2345–2349. [CrossRef]
49. Bakumenko, A.; Hlaváčková-Schindler, K.; Plant, C.; Hubig, N.C. Advancing anomaly detection: Non-semantic financial data encoding with LLMs. *arXiv* **2024**, arXiv:2406.03614.
50. Zhu, X.; Jiang, L.; Gao, Y.; Yin, Y. Research on financial statement analysis methods based on machine learning. In Proceedings of the 2023 3rd Guangdong-Hong Kong-Macao Greater Bay Area Artificial Intelligence and Big Data Forum, Guangzhou, China, 22–24 September 2023; pp. 31–36.
51. Wu, J.; Lu, W.; Yan, G.; Li, X. MLA: Machine learning adaptation for realtime streaming financial applications. In Proceedings of the 2019 Tenth International Green and Sustainable Computing Conference (IGSC), Alexandria, VA, USA, 21–24 October 2019; pp. 1–6. [CrossRef]
52. Polytarchos, E.; Bardaki, C.; Pramatari, K. Assessment of the real-time pattern recognition capability of machine learning algorithms. *Stat. Anal. Data Min. Asa Data Sci. J.* **2024**, *17*, e11701. [CrossRef]
53. Kyriakopoulos, F.; Thurner, S.; Puhr, C.; Schmitz, S.W. Network and eigenvalue analysis of financial transaction networks. *Eur. Phys. J. B* **2009**, *71*, 523–531. [CrossRef]
54. Gao, W.; Su, C. Analysis on block chain financial transaction under artificial neural network of deep learning. *J. Comput. Appl. Math.* **2020**, *380*, 112991. [CrossRef]
55. Malinowski, A.; Schlather, M.; Zhang, Z. Marked point process adjusted tail dependence analysis for high-frequency financial data. *Stat. Its Interface* **2015**, *8*, 109–122. [CrossRef]
56. Hautsch, N.; Pohlmeier, W. *Econometric Analysis of Financial Transaction Data: Pitfalls and Opportunities*; University of Konstanz, Center of Finance and Econometrics (CoFE): Konstanz, Germany, 2001.
57. Sukharev, O. Investments in the transaction sector and financial assets: Impact on economic growth. *Financ. Theory Pract.* **2020**, *24*, 60–80. [CrossRef]
58. Krishnapriya, G. Identification of Money Laundering based on Financial Action Task Force Using Transaction Flow Analysis System. *Bonfring Int. J. Ind. Eng. Manag. Sci.* **2017**, *7*, 1–4. [CrossRef]
59. Wang, Z. Abnormal financial transaction detection via ai technology. *Int. J. Distrib. Syst. Technol. (IJDST)* **2021**, *12*, 24–34. [CrossRef]
60. Wiryadinata, D.; Sugiharto, A. The Use of Machine Learning to Detect Financial Transaction Fraud: Multiple Benford Law Model for Auditors. *J. Inf. Syst. Eng. Bus. Intell.* **2023**, *9*, 239–252. [CrossRef]
61. Leskovec, J.; Lang, K.; Dasgupta, A.; Mahoney, M.W. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Math.* **2009**, *6*, 29–123. [CrossRef]
62. Weber, J.; Sayre, R.; Shi, C.; Gibbs, J.P.; Paxson, V. Anti-Money Laundering Simulator. 2018. Available online: https://arxiv.org/abs/1812.00076 (accessed on 16 October 2024).
63. Altman, E.; Blanuša, J.; Von Niederhäusern, L.; Egressy, B.; Anghel, A.; Atasu, K. Realistic synthetic financial transactions for anti-money laundering models. In Proceedings of the Advances in Neural Information Processing Systems, Orleans, LA, USA, 10–16 December 2023; Volume 36.
64. Karras, A.; Giannaros, A.; Theodorakopoulos, L.; Krimpas, G.A.; Kalogeratos, G.; Karras, C.; Sioutas, S. FLIBD: A federated learning-based IoT big data management approach for privacy-preserving over Apache Spark with FATE. *Electronics* **2023**, *12*, 4633. [CrossRef]
65. Karras, A.; Giannaros, A.; Karras, C.; Theodorakopoulos, L.; Mammassis, C.S.; Krimpas, G.A.; Sioutas, S. TinyML algorithms for big data management in large-scale IoT systems. *Future Internet* **2024**, *16*, 42. [CrossRef]
66. Karras, A.; Karras, C.; Schizas, N.; Avlonitis, M.; Sioutas, S. AutoML with Bayesian optimizations for big data management. *Information* **2023**, *14*, 223. [CrossRef]

67. Schizas, N.; Karras, A.; Karras, C.; Sioutas, S. TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. *Future Internet* **2022**, *14*, 363. [CrossRef]
68. Karras, C.; Theodorakopoulos, L.; Karras, A.; Krimpas, G.A. Efficient algorithms for range mode queries in the big data era. *Information* **2024**, *15*, 450. [CrossRef]