IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 20 June 2024, accepted 24 July 2024, date of publication 30 July 2024, date of current version 8 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3435705

RESEARCH ARTICLE

Enhancing Task Management in Apache Spark Through Energy-Efficient Data Segregation and Time-Based Scheduling

NADA M. MIRZA[1,2], ADNAN ALI[3], NURA SHIFA MUSA[4], AND MOHAMAD KHAIRI ISHAK[5], (Member, IEEE)

[1]School of Electrical and Electronic Engineering, Universiti Sains Malaysia (USM), Nibong Tebal, Pulau Pinang 14300, Malaysia
[2]College of Engineering, United Arab Emirates University, Al Ain, United Arab Emirates
[3]Information Technology Center, University of Khorfakkan, Khorfakkan, United Arab Emirates
[4]College of Engineering, Al Ain University, Al Ain, United Arab Emirates
[5]Department of Electrical and Computer Engineering, College of Engineering and Information Technology, Ajman University, Ajman, United Arab Emirates

Corresponding author: Mohamad Khairi Ishak (m.ishak@ajman.ac.ae)

This work was supported in part by Ajman University, United Arab Emirates.

ABSTRACT The rise of smart cities as solutions to urban challenges has garnered significant attention in recent years. With technological advancements, particularly in wireless communication and artificial intelligence, smart cities aim to optimize decision-making processes and improve citizen services. This study explores the integration of extensive infrastructure and networked Internet of Things (IoT) devices to collect data and enhance city performance. With urban populations steadily increasing, the need for efficient resource management and sustainability practices becomes paramount. However, challenges such as energy trading, privacy concerns, and security issues persist. To address these challenges, big data analytics (BDA) systems are crucial, necessitating efficient task scheduling strategies. This study proposes a Dynamic Smart Flow Scheduler (DSFS) system for Apache Spark, showcasing significant improvements in resource efficiency and task optimization. By reducing resource consumption and task execution, the proposed approach enhances system performance, scalability, and sustainability.

INDEX TERMS Apache spark, data segregation, energy efficient, smart cities, dynamic scheduler.

I. INTRODUCTION

The growth of urban populations and the increasing complexity of cities pose significant challenges in transportation, environmental sustainability, energy management, and social organization. The concept of smart cities has appeared as a technological solution to address these issues. While similar ideas have been around for over a decade, recent technological advancements, such as wireless communication and artificial intelligence, have significantly enhanced the capabilities and diversity of smart city applications. As a result, smart cities are poised to become profitable business opportunities soon [1], [2].

The smart city concept involves the integration of extensive infrastructure, incorporating networked Internet of Things (IoT) devices that collect data from the physical environment. The primary objective is to optimize decision-making processes to enhance performance and deliver improved services to citizens. The development of smart city services presents a complex challenge in which technology is experiencing exponential growth. These services require optimization based on various criteria. To strengthen city administration, enhance city services, and alleviate city functions, countries including the United States, Britain, Japan, and Korea are focusing on the next generation of the internet and speeding up the implementation of smart city systems [3]. More than half of all humans already live in urban areas, and that number is anticipated to go over five billion by 2030 [4]. According to the latest projections by the United Nations, by 2050,

The associate editor coordinating the review of this manuscript and approving it for publication was Yuan Gao.

a significant number, 68% of the world's population, is predicted to live in cities. These numbers mark a significant shift from the 751 million people living in urban regions in 1950 to an impressive 4.2 billion individuals in 2018 [4]. This rapid urbanization trend has come with its challenges. Researchers Bibri and Krogstie [5] have highlighted that urban areas account for an overwhelming 70% of total natural resource consumption, contributing to environmental degradation, the destruction of ecosystems, and energy shortages. A big problem for cities is needing more important resources such as water and money.

Similarly, the networking between buildings, where information, technology, and communication play a significant role, is central to the smart city concept [6]. Smart cities have a long way to go before fully addressing concerns such as energy trading, agent privacy, and security. Providing people with clean water, a reliable food supply, and enough energy to meet their needs while maintaining economic, social, and environmental sustainability is an increasing issue [7], [8].

As smart cities continue to grow, they generate massive amounts of data, often referred to as big data. This data will be crucial for the functioning of services connected to the IoT. Big data has certain characteristics, such as being very large, moving very fast, and coming in various types, which researchers have extensively studied. Smart cities can gain valuable insights from this diverse data, even though much of it needs to be neatly organized like data collected in other ways. Smart cities can use cloud platforms or data centers with strong, reliable databases to manage this huge amount of unstructured data efficiently. Special programming techniques that can process and analyze huge numbers of data at once can be used to add up all the collected data from multiple sources in smart cities. This phenomenon is called big data analytics (BDA), and while it is still relatively new regarding smart cities, it has a great ability to improve urban services.

In a BDA system, many tasks need to be done simultaneously on a group of different computers. However, ensuring these tasks are scheduled efficiently is a significant challenge and can slow down the system. To enhance scheduling, researchers have developed various strategies. Some, such as Sparrow, KMN, Orchestra, and Apollo, have created systems that distribute the task scheduling workload or reduce the amount of data that needs to be moved around. These strategies prove useful in systems such as Hadoop and Spark, which are employed for handling big data [8].

Researchers have focused on improving task scheduling in computer systems such as MapReduce, concentrating on three main aspects: making scheduling smarter based on how fast computers can perform tasks, reducing the amount of data sent between computers during scheduling, and enhancing task efficiency. Most of this work has been done for Hadoop, with limited attention given to Spark. This emphasizes the importance of improving Spark's scheduling as well.

Spark differs from Hadoop in that it prioritizes memory usage, significantly enhancing processing speed. Memory plays a crucial role in determining Spark's efficiency. Moreover, Spark utilizes a distinct task-handling approach, employing one thread per task compared to Hadoop's use of multiple processes. Considering Spark's distinctive characteristics and drawing insights from existing practices, this study proposes a new task-scheduling approach for Spark.

- The work introduces a method for data segregation through the categorization of information based on its source stream. This approach ensures a highly organized and efficiently managed data structure.
- The system has been designed to save energy and time more effectively by implementing a sleep mode at specific active times during sleep hours, leading to significant savings in energy and time.
- The introduction of a time-based scheduling system is an important contribution to streamlining task management. This system allows for a more precise and efficient allocation of resources and workflows, thereby improving overall task management effectiveness.

The proposed Dynamic Smart Flow Scheduler (DSFS) system makes a robust contribution by simplifying city administration and improving residents' lives. Additionally, it collects valuable user data, serving as a foundational resource for government and decision-making entities in planning, resource allocation, research, and analysis.

II. SMART CITY CHARACTERISTICS

A smart city represents a fundamental change in how urban areas are developed and managed, focusing on the efficient use of resources. This efficiency encompasses tangible elements such as transportation and energy systems, as well as intangible assets such as human expertise and administrative skills, all managed in real-time. For instance, consider a smart traffic system that uses real-time information to improve traffic flow and reduce fuel consumption. Moreover, smart cities are equipped to respond effectively to crises such as floods, fires, and earthquakes, enabling emergency response, disaster relief, and self-automated help in hazardous areas. Another concept is to reduce the usage of nonrenewable resources as they will eventually run out, unlike renewable resources such as solar, wind, and geothermal energy, which can be sustained indefinitely. To address this reality, there is a growing emphasis on adopting smart energy, green energy, and sustainable energy practices to raise awareness about energy challenges and promote optimal energy consumption strategies. Characterizing a smart city involves several key attributes that are essential for creating a technologically advanced urban environment. Examples such as smart traffic systems, the utilization of renewable energy sources, and efficient energy grids help make the idea of smart cities easy to understand.

A. SMART HEALTHCARE

The increasing global population is exerting immense pressure on healthcare systems worldwide. Compounding this challenge is the insufficient number of medical professionals to meet the escalating demand, creating a widening gap between healthcare needs and available services. Limited resources further exacerbate the situation. To address this issue, modern healthcare can leverage technology, including sensor networks, information and communication technology, cloud computing, mobile apps, and robust data processing systems [9]. A practical example is the use of electronic clinical records, which are digital records of a person's health information. These records assist doctors in making informed decisions by providing the most recent patient information, thereby enhancing overall care quality. Mobile health solutions, such as health-related apps on smartphones, contribute to making healthcare more accessible and convenient. For example, an app that monitors health conditions or provides quick advice from a healthcare professional without clinic visits enhances efficiency and responsiveness, ultimately improving well-being in urban areas [10].

B. SMART TRANSPORTATION

Smart transportation is a fundamental component of smart cities, addressing issues associated with road traffic and impacting living standards and safety—the primary goals of smart city development. Technologies such as the IoT, GPS, wireless technologies, and sensing technologies enable smart transportation. As these applications deal with dynamic and heterogeneous devices and data, they are inherently broad and complex. Common features include intelligent parking systems, public transit systems, traffic management, taxi applications, traffic signal systems, and disaster prevention systems—scheduling techniques for smart transportation focus on optimizing traffic flow and resource allocation. Dynamic routing for shared transportation services, such as shared vehicles and bicycles, can enhance travel efficiency, reduce congestion, and contribute to a more accessible transportation system [11].

C. SMART WASTE MANAGEMENT

Urbanization and increased manufacturing activities contribute to excessive waste production. Effective waste management requires collaboration between labor forces, municipal entities, and private enterprises. This complex process includes waste collection, removal, reuse, and recovery. IoT technology aids smart waste management by using sensors to identify overflowing bins and notify authorities. Incorporating a scheduling strategy enhances the efficiency of smart waste management systems [12].

D. SMART INDUSTRY/MANUFACTURING

The intersection of industry and technology has ushered in a new era of urban development and smart cities characterized by unprecedented efficiency, sustainability, and connectivity. Scheduling in smart industries utilizes sensor data to predict maintenance needs and proactively schedule activities, minimizing downtime and optimizing production efficiency. Additionally, scheduling techniques can optimize energy consumption by aligning energy-intensive tasks with periods of low demand or abundant renewable energy sources [13].

III. RELATED WORKS

The vast reservoirs of big data within smart cities have facilitated transformative advancements across various sectors. The intricate web of interconnected devices and sensors capturing real-time information creates opportunities for enhanced scheduling and resource management. The collaboration between BDA and efficient scheduling mechanisms has led to innovative approaches optimizing everything from transportation networks to utility distribution. This interplay underscores the pivotal role of information technology in shaping the future of urban planning and resource allocation. Harnessing the power of big data empowers decision-makers with an understanding of city dynamics. It facilitates the design and implementation of agile scheduling frameworks, maximizing efficiency and responsiveness across the smart city ecosystem [14], [15].

Significant strides have been made in recent studies, digging into the potential of data management and analytics in optimizing decision-making processes in smart cities. These studies underscore the crucial role of integrating extensive infrastructure and networked Internet of Things (IoT) devices to collect data and enhance city performance [16], [17]. As urban populations continue to grow, the need for efficient resource management and sustainability practices becomes more pronounced. However, challenges such as energy trading, privacy concerns, and security issues persist within Smart Grids, a pivotal component of smart cities. Big data analytics (BDA) systems are identified as a key tool for tackling these challenges in Smart Grids and broader smart city initiatives. Yet, their effectiveness is contingent on efficient task-scheduling strategies, particularly in resource-constrained environments. To overcome this gap, several research efforts have explored the challenges and opportunities associated with data management and security in Smart Grids. The increasing volume of data exchanged between devices and open communication channels raises concerns about data privacy and security. To address these vulnerabilities, researchers have proposed leveraging blockchain technology for secure and transparent peer-to-peer energy trading. Smart contracts on blockchains can automate energy trading processes and guarantee secure transactions [18], [19], [20], [21], [22].

Numerous scheduling algorithms have been developed to address fundamental challenges associated with task scheduling, employing various techniques and strategies to optimize data locality, synchronize processing, and reduce task completion times.

The Longest Approximate Time to End (LATE) [23] scheduler utilizes backup tasks for those with prolonged execution times. By assigning fixed weights to estimate remaining execution time, LATE identifies tasks with slower execution rates and redistributes them to alternative nodes. This approach optimizes task execution times based on precise calculations of remaining execution time and job progress rates. However, inaccuracies in estimations may lead to incorrect task selections for re-execution. In response, Chen et al. introduced the Self-Adaptive MapReduce (SAMR) [24] algorithm, leveraging historical context to enhance the accuracy of remaining execution time estimations. SAMR improves efficiency by considering execution times and system resource distribution, prioritizing data locality.

The Combination Re-Execution Scheduling Technology (CREST) [25], proposed by Lei et al., aims to minimize response times for MapReduce jobs and optimize speculative map task performance by re-executing combinations of tasks on specific computing nodes, capitalizing on data locality. Conversely, Hammoud and Sakr [26] presented the Locality-Aware Reduce Task Scheduler (LARTS), tailored to address data locality concerns specific to reducing tasks and optimizing data locality by utilizing network location information and partition sizes.

Ibrahim et al. [27] introduced the Maestro scheduling algorithm to mitigate non-local map task execution issues by incorporating replica-aware execution. Maestro minimizes the impact of local map task execution on other nodes, optimizing task mapping and enhancing stable data distribution during the shuffling phase.

The Matchmaking-scheduling algorithm by He et al. [28] prioritizes local map tasks using a locality marker, ensuring equitable distribution among nodes. Another algorithm, named HybS, was introduced by Le et al. [29], operates on dynamic priority principles, reducing delays in concurrent jobs and maintaining data locality, offering a user-defined quality of service levels. Gounaris et al. [30] approach investigates running Spark applications on a local machine using a standalone cluster environment with first-in-first-out scheduling. While acknowledging the benefits of cloud computing, this research explores a local setup proposing first-in-first-out for its simplicity and potential to improve cloud efficiency and reduce costs. The research analyzes execution time and resource utilization. Javanmardi et al. [31] propose a high-level architecture for job scheduling that prioritizes factors like data locality, fairness, performance optimization, and automation. The architecture discussed in their research leverages a scheduler with dedicated components for job execution and data distribution across nodes in a heterogeneous Hadoop cluster. The high-level architecture addresses challenges like scheduler program size, execution time estimation, and cluster heterogeneity.

Zhang et al. [32] propose two techniques for Spark job scheduling: Rate Monotonic Decreasing Utilization for Tasks. This algorithm focuses on task-level scheduling and aims to minimize resource usage. The second technique is Periodic Task-oriented Scheduling for job-level scheduling, which focuses on improving execution efficiency by assigning sub-jobs effectively within available resources.

Chen et al. [33] introduce a Parallel Random Forest algorithm for big data processing using Apache Spark. This algorithm leverages a hybrid approach that combines data-parallel and task-parallel optimization to improve performance within the Spark computing environment. Another research conducted by Verma et al. [34] highlights the challenges of scheduling Spark jobs in cloud environments. It discusses the limitations of existing techniques in heterogeneous cloud environments with various constraints and the potential of reinforcement learning algorithms to optimize resource utilization by analyzing job scheduling. The case study demonstrates how this approach can enable an agent to learn the inherent characteristics of the computing environment for improved scheduling.

In 2022, Tang et al. [35] conducted a survey that explores memory management techniques crucial for in-memory computing systems. Many data processing frameworks rely on garbage-collected languages like Java, which can introduce performance overhead due to garbage-collection pauses. Studies address this issue by either improving garbage collector performance or leveraging application semantics to manage memory explicitly and eliminate this overhead. One approach focuses on coordinating garbage collection across different nodes where Spark workloads run. This eliminates situations where all nodes must wait for a single node to experience a garbage collection pause. Alternatively, Spark itself can manage memory to avoid garbage collection overhead. Techniques like Tungsten leverage off-heap memory and utilize code generation to optimize performance and memory usage. Table 1 below provides a brief overview of select scheduling algorithms.

Setting resource requirements for Spark can be challenging due to its numerous configuration parameters. Researchers have fine-tuned parameters to enhance system performance [41], while Gounaris et al. addressed resource wastage when Spark applications utilize all nodes. Gibilisco et al. [42] developed regression models predicting application execution times based on profile data. Wang and Khan [43] focused on modeling performance in Directed Acyclic Graph-based in-memory analytics platforms. Islam et al. [44] introduced an approach with a guaranteed deadline for Spark jobs allocation into fine-grained resources. Delimitrou and Kozyrakis [45] and Jyothi [46] use classification techniques to analyze resource impacts and dynamically adjust allocations, while Morpheus also optimizes cluster performance by resolving failed job issues. Dimopoulos [47] operates as a fair-share resource allocator, adapting to workload changes and ensuring sufficient resources for tasks to meet deadlines promptly. Several other studies have also been proposed to improve micro-batch stream job scheduling within Apache

TABLE 1. Scheduling algorithms overview.

Scheduling Algorithm	Pros	Cons
LATE [23]	It utilizes backup tasks for prolonged execution times.	The remaining time estimations may need to be more accurate.
SAMR [24]	It uses historical context for accurate estimations.	Complex to implement.
CREST [25]	It minimizes response times for MapReduce jobs.	Task combinations can complicate the execution.
LARTS [26]	It optimizes data locality for reducing tasks.	It is limited to specific tasks.
Maestro [27]	Mitigates non-local map task issues.	Complexity increases with larger tasks.
Matchmaking [28]	It prioritizes local map tasks.	It may lead to uneven load distribution.
HybS [29]	It reduces delays in concurrent jobs.	Quality of service levels may vary.
Data locality-driven [36][36]	It optimises for heterogeneous clusters.	Complexity in managing varying capacities.
HDFS data layout scheme [37]	Optimises data distribution and replication.	Complexity in dynamic allocation.
Communication cost models [38]	Minimize communication costs using graphs.	It requires accurate modeling of interactions.
Map splits distribution [39]	Addresses challenges of distributed map splits.	Limited impact on overall scheduling.
Data-locality-aware [40]	Optimises makespan with data-locality awareness.	It may only be effective in some scenarios.

Spark, which introduced the concept of dividing a live data stream into micro-batches for parallel processing [48]. Effectively scheduling of small batches of data to gain low latency and high performance can be challenging. One more approach is the A-scheduler, an adaptive scheduling algorithm proposed by Cheng et al. [49] algorithm dynamically assigns schedules to parallel small batches of jobs based on data dependencies. It utilizes two job pools, which can be categorized as dependent and independent. The independent pool uses a priority-based sharing policy, while the dependent pool follows a first-in, first-out approach. Another approach is the data-driven priority scheduler presented by Ajila and Majumdar [50]. This scheduler also allows users to define priorities for all types of input data, guaranteeing timely execution of items with higher priority even during peak load time. Additionally, it can be used alongside the techniques working on dynamic resource allocation principles. Then, in 2019, Garefalakis et al. [51] introduced Neptune, which is a framework for unified stream and batch applications. It implements a locality- and memory-aware scheduling policy to determine which tasks to suspend and when. Similarly, the Multilevel Dynamic Feedback Scheduling (MDFS) algorithm developed by Natarajan and Subramanian [52] aims to minimize transmission delay and increase the delivery rate. The time quantum determines a packet priority for which the model provides three levels, the highest being urgent data. Priority management is supported at each node by a system of three queues connected by a feedback mechanism. The MDFS performs inter-queue migrations based on the comparison of the time quantum of the packet to the minimum and maximum limit values of the queue to avoid starvation problems

IV. PROPOSED SCHEDULING SYSTEM

Spark's scheduling dynamically manages tasks across a cluster, optimizing resource allocation and sequencing. In-memory processing minimizes data shuffling, enhancing efficiency and speed. This integrated scheduling maximizes computation overlap and minimizes data movement for faster performance. Essentially, the shift from conventional scheduling algorithms to frameworks such as Spark exhibits the evolution of data processing. These modern frameworks combine insights from past strategies with in-memory processing and sophisticated optimizations for agile, high-performance data processing.

This study introduces a unique framework with multilayer, dynamic, priority-based, and time-sensitive data processing and scheduling. Multilayer refers to the hierarchical structure of data processing with multiple levels or layers for effective resource management. Setting task priorities in a priority-based manner guarantees that crucial processes are given more urgency to improve system performance. Dynamic refers to a system's ability to change in real-time to changing situations, such as shifting workloads and resource availability. Lastly, time sensitivity highlights the temporal component, emphasizing the necessity of completing duties promptly and within designated timeframes. When combined, these qualities provide an intricate structure that prioritizes tasks, emphasizes timely execution, and provides an organized hierarchy to maximize data processing. This approach aims to elevate the efficiency and responsiveness of data processing and scheduling in dynamic computing environments.

The proposed system is designed to intelligently manage and schedule data streams originating from various sources within a smart city. It leverages Python and Spark, two powerful technologies for data processing and analysis. The core functionality of this system revolves around assigning priority levels to different types of data streams and efficiently managing their transmission and processing. It works within the context of the proposed multilayer network architecture designed for smart city applications.

In the proof-of-concept scenario outlined in the paper, the transfer of files or any additional overhead associated with network resource utilization is not included. The focus of this paper is specifically on demonstrating the effectiveness of the proposed methodology in dynamically allocating resources and improving data processing efficiency within a cluster environment. By concentrating on elements closely related to the core concept of the paper, such as monitoring resource utilization, dynamically activating additional nodes, and optimizing task allocation, the main aim is to provide

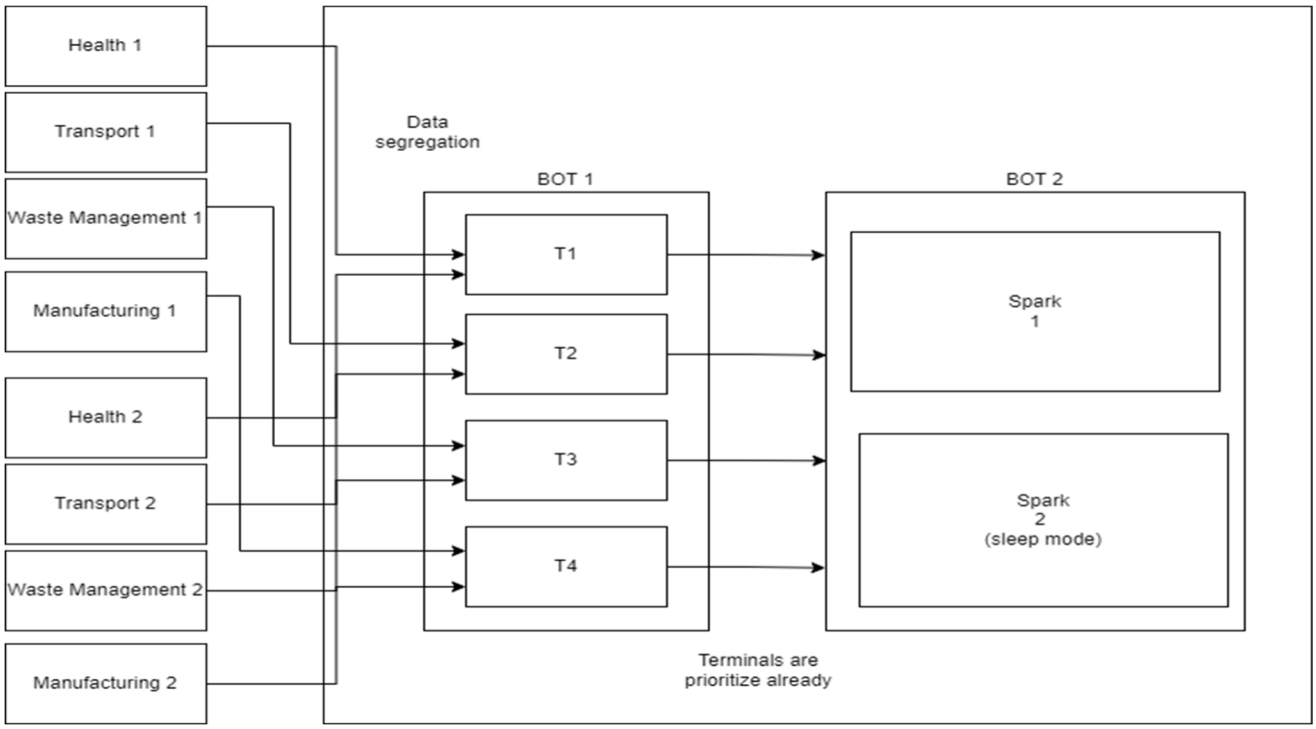

FIGURE 1. System flow chart.

a clear and focused evaluation of the proposed approach. This decision allowed for streamlining of the proof-of-concept implementation and focused analysis on the key aspects relevant to demonstrating the feasibility and effectiveness of the proposed methodology within the scope of the paper.

The system flowchart is depicted in Figure 1.

The following components are implemented in the designed system:

A. A RESOURCE-BASED

The proposed methodology employs Python's robust libraries to establish remote connectivity and retrieve real-time system resource utilization. Paramiko, a Python library, enables secure remote access to nodes, allowing seamless activation and deactivation based on demand. Psutil, a cross-platform library, retrieves information about running processes and system utilization.

The approach addresses several limitations of traditional Spark setups by leveraging these libraries. To avoid the energy inefficiency of keeping all nodes active during low-demand periods, the system dynamically activates and deactivates the second node based on real-time load using Python scripts, significantly reducing energy consumption. Unlike traditional Spark, which lacks real-time dynamic resource allocation capabilities, the proposed system continuously monitors the first node's load using Psutil. The system adapts resource allocation, accordingly, ensuring efficient resource utilization and better handling of workload spikes.

To address the potential load imbalance in traditional Spark setups, which may lead to the overloading of some nodes, the proposed approach dynamically redistributes tasks based on real-time load data, preventing overloading and ensuring optimal performance. Traditional Spark setups often require manual intervention for scaling, leading to inefficiencies. This approach overcomes this limitation by automatically scaling resources based on real-time system metrics collected by Psutil. Paramiko then facilitates secure remote execution of commands to manage these scaling actions.

By integrating Paramiko and Psutil, the system achieves a more efficient, flexible, and easily manageable Spark cluster. It adapts to changing workload demands, eliminating issues like energy inefficiency, lack of real-time adaptation, load imbalance, and manual scaling.

A carefully chosen 70% resource usage threshold ensures a sufficient buffer for workload spikes while avoiding unnecessary activations of additional resources. This threshold balances performance and energy efficiency, activating extra processing power only when there's a sustained need. The wake-up signal tells the second machine to switch from a low-power mode, like sleep or hibernation, to an active state. Normally, the second machine stays in a low-power mode to save energy until it gets the wake-up signal. This method ensures that extra resources are only used when they're really needed. By sending the wake-up signal at the 70% threshold, the system maintains a balance between resource use and system responsiveness, keeping some resources in reserve for workload spikes without overloading the system. Using the

wake-up signal helps manage resources efficiently, allowing the system to adjust resources based on demand while saving energy during quiet periods.

B. PRIORITY-DRIVEN DATA MANAGEMENT

A key factor contributing to the effectiveness of the proposed approach is its commitment to a well-defined system that prioritizes data transmission based on a carefully established hierarchy of importance. This foundational principle ensures that data with higher priority is consistently given preference over lower-priority data when arranging data for transmission. By incorporating this priority-based framework into its operational design, the proposed approach optimizes the sequence in which data is sent, facilitating the rapid and efficient distribution of critical information. The approach stands out by efficiently organizing data buffers to enhance data processing and scheduling, especially when handling data from multiple terminals. These streams of data are divided into four priority levels:

Healthcare Data (Priority Level 0.4): This classification is dedicated to the most critical data, notably healthcare and public safety information. It guarantees immediate transmission and prioritizes vital data, fostering a remarkably responsive and efficient data management system.

Smart Transportation (Priority Level 0.3): Data related to smart transportation is assigned a priority level of 0.3, signifying its importance and ensuring timely processing and transmission.

Waste Management (Priority Level 0.2): Data associated with waste management is given a priority level of 0.2, indicating its significance in efficient waste handling and resource management.

Sustainability / Manufacturing / Industry 4.0 (Priority Level 0.1): Data related to sustainability, manufacturing, and Industry 4.0 is assigned a priority level of 0.1. This level acknowledges its importance and ensures it is processed and transmitted efficiently within the system.

The prioritization of tasks such as healthcare data, smart transportation, and waste management in the city is determined based on their urgency and impact on city operations. It is crucial to prioritize tasks in detail, as specific tasks within each area can have varying levels of urgency. For example, an urgent transportation task must take precedence over a healthcare task at a given moment.

To address this, each task is assigned an L1 (priority), an L2 (number of files), and an L3 (file size). The aOrder of Processing is calculated using the formula L1 * (L2 + L3). This formula effectively balances priority with task size and demand, ensuring that tasks are prioritized based on both immediate demand and their inherent complexity and scale. By considering the priority (L1) and the workload metrics (L2 and L3), our approach effectively manages and sequences tasks to optimize performance and responsiveness.

C. TIME-DRIVEN DATA MANAGEMENT

Low-priority tasks, while occupying on-disk and in-memory space, are managed through several resource optimization techniques. These include data compression and efficient memory management to minimize resource usage. Our scheduling algorithms ensure that resources are periodically allocated to lower-priority tasks, preventing starvation and ensuring all tasks are eventually processed. Specifically, at intervals of 30 minutes, the system assesses inactive terminals to determine if any actions are required. If a terminal is found to be unused during this check, relevant procedures and tasks are initiated. This systematic monitoring ensures that resources are managed efficiently, and actions are taken only, when necessary, based on observed usage patterns, thereby optimizing overall resource utilization and maintaining system performance.

D. ALGORITHM FLOWCHART

Figure 2 shows the algorithm flowchart to explain how the DSFS approach works.

The following is a step-by-step plan that explains all the important decisions it makes. The file-processing workflow within the computing environment follows a systematic and efficient approach to handling segregated files. The process begins by retrieving files from the terminal, ensuring a well-organized structure for subsequent tasks. Priority levels are then calculated for each file, establishing a predetermined order for processing.

An initial check is performed on the resources available on the first machine. If the machine possesses the necessary resources, the files are processed accordingly. However, if the primary machine lacks the required capacity, a series of steps are initiated to redistribute the workload.

Firstly, the second machine is activated to handle the additional processing load. The designated files are then seamlessly transferred to the second machine for efficient handling. Upon completion of processing on the second machine, a signal is sent back to the main system to indicate the successful execution of tasks.

Once the processing is finalized, the system proceeds to post-processing actions. This involves checking for newly arrived files and assessing the availability of resources on the first machine. If new files are detected and resources are available on the primary machine, the system seamlessly processes these files. Subsequently, a sleep signal is dispatched to the second machine, conserving energy when it is not actively engaged in processing tasks.

This entire file-processing cycle operates iteratively to ensure a continuous and automated workflow. The system regularly checks for new files, evaluates resource availability, and optimizes processing on the available machines. This structured approach not only streamlines the processing of files but also maximizes the utilization of resources within the computing environment. By dynamically allocating tasks based on resource availability, the system adapts to varying

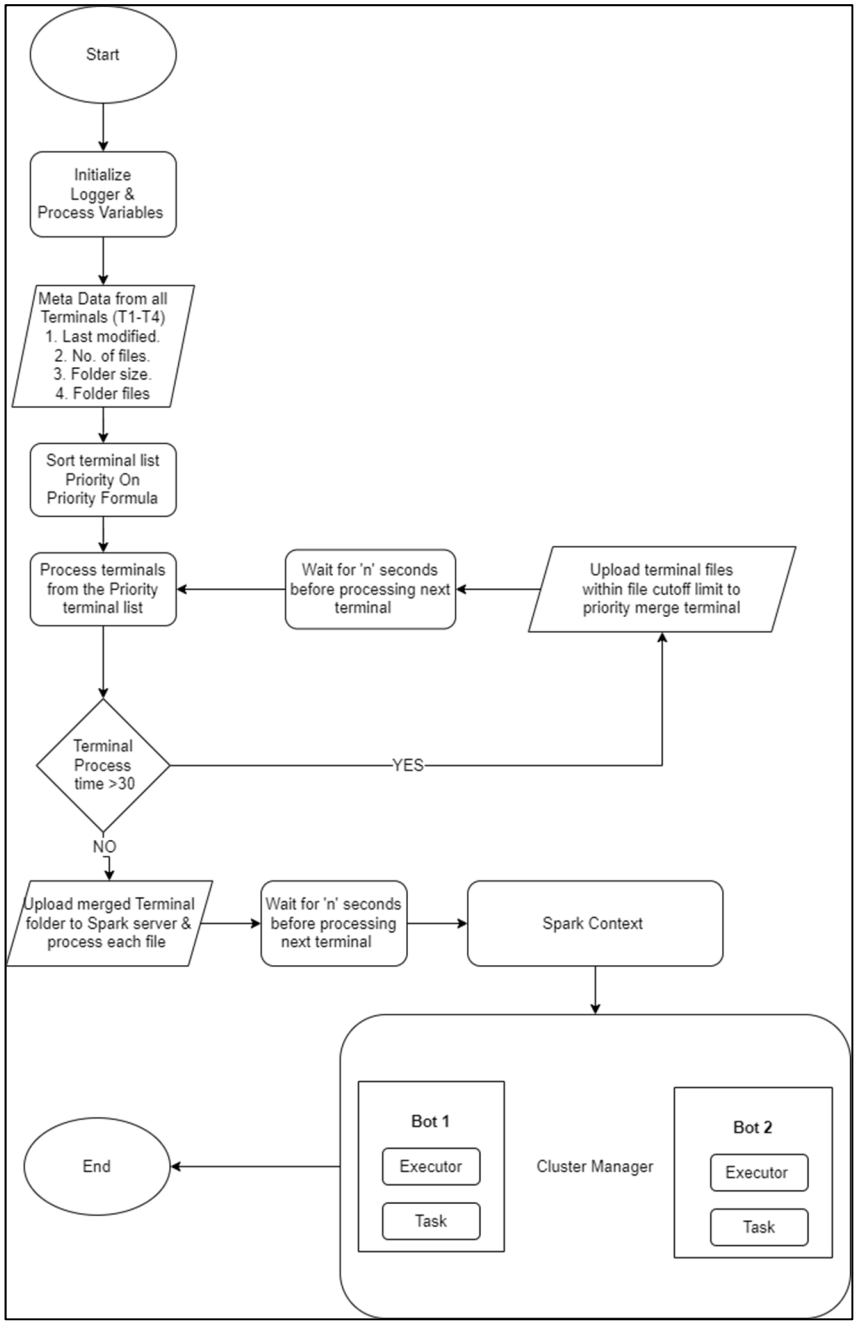

FIGURE 2. Algorithm flow chart.

workloads, maintaining efficiency and productivity in the file-processing workflow.

V. RESULT

This section discusses the proposed system's results and findings, explaining how the order of operation for each terminal is determined based on its priority, the number of files, and the size of the files.

Furthermore, the mathematical relationship is also discussed to help understand how the proposed system saves energy during its operations.

Let us assume the following:

Number of files in traditional approaches $= NF_o$

The number of files in the proposed approach $= NF_m$

Time taken by traditional approaches $= t_o$

Time is taken by the proposed approach $= t_m$

Average file size = Sum of all file sizes / total number of files $= (S_t/S)$

As per the conditions applied in the proposed approach, it can be easily stated that:

$$NF_{m=}NF_o - (S_t/S) * 0.05 \quad (1)$$

This equation leads to the fact that:

$$NF_m < NF_o \tag{2}$$

Knowingly, the time taken is directly proportional to the number of files.

$$t_o \propto NF_o \tag{3}$$

$$t_m \propto NF_m \tag{4}$$

By comparing Eq. (2) with Eq. (3) and Eq. (4) it can be easily stated that

$$t_m < t_0 \tag{5}$$

Similarly,
Servers in BOT 2 are named as N_{1b}, N_{2b}
The energy utilized by each server $= E_b$
The energy utilized by the traditional approach $=$

$$E_o = N_{1b} * E_{1b} + N_{2b} * E_{2b} \tag{6}$$

In the proposed approach, all servers will never be used completely; initially, the first server will be utilized to 80%, and then the files will be transferred to the next one while trying to keep the energy utilization in the second server only up to 20%. The choice of an 80% utilization threshold is based on balancing resource efficiency and ensuring optimal system performance. This threshold allows the server to operate at a high level of utilization, maximizing resource efficiency while still providing some buffer to accommodate workload spikes and ensure responsiveness. By setting an appropriate utilization threshold based on these factors, the system can effectively manage workload demands, maintain stability, and optimize resource utilization, ultimately enhancing overall system efficiency and performance. Therefore, as per the given conditions, the energy utilized for the proposed approach will be:

$$E_m = N_{1b} * 0.8E_{1b} + N_{2b} * 0.2E_{2b} \tag{7}$$

By comparing (6) and (7), it can be easily concluded that the proposed approach is more energy-efficient than traditional approaches. Furthermore, distributing the workload across multiple servers offers several advantages over relying solely on a single server at 100% capacity. By splitting the workload, resources are allocated more efficiently, with Server 1 operating at 80% utilization and Server 2 at 20%. This ensures that neither server is overburdened, reducing the risk of performance issues or resource contention.

Utilizing multiple servers also provides fault tolerance and redundancy. If a hardware failure or system problem occurs on one server, the system can isolate the fault and continue operations on the remaining server(s), minimizing downtime. The redundancy of multiple servers enhances the overall reliability of the system.

Furthermore, distributing the workload across multiple servers offers greater scalability and flexibility. As workload demands change or the system needs to accommodate growth, additional servers can be added to handle the increased load, which is difficult to achieve with a single server at full capacity. The ability to optimize utilization percentages across multiple servers allows the system to adapt to varying needs over time.

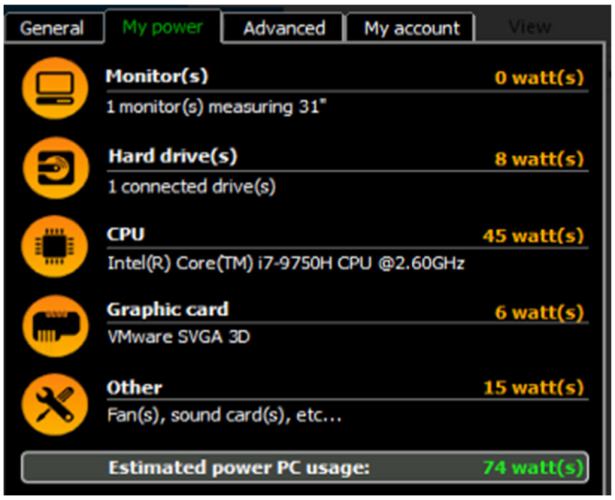

FIGURE 3. System specifications.

To generate results, the main system and two spark servers are used; each server has 6 GB RAM, 4 GB storage, and four cores each (VM), and each server utilizes around 74 watts, as given below in Figure 3.

The following test cases are run into the proposed approach:

In Table 2, four terminals (T1–T4) are presented along with their associated parameters. Each terminal was assigned an L1 (priority), an L2 (number of files), and an L3 (file size). The calculated value given in the ''Order of Processing'' column indicates the sequence in which the terminals were processed based on their priority and workload metrics, derived from the generic formula (L1 * (L2 + L3)).

The formula L1*(L2 + L3) for determining the execution sequence is derived from a consideration of both workload demand and task complexity. The term L1 represents the priority. By multiplying this with the sum of L2 and L3, which signify the file size and the number of files, respectively, the formula effectively balances priority with task size and demand. This ensures that tasks are prioritized based not only on immediate demand but also on their inherent complexity and scale. Larger files or a greater number of files inherently require more computational resources and time to process, making them higher-priority tasks. Considering file size and the number of files in task prioritization allows for intelligent resource allocation, ensuring optimal utilization and preventing overloading of nodes with resource-intensive tasks. Ultimately, this approach enhances system efficiency, performance, and responsiveness in data processing operations, leading to more effective utilization of computational resources and improved overall system performance.

In Case 01, the order of processing is determined by the calculated values, resulting in T4 being processed first with

TABLE 2. Test cases.

Terminal	L1-Priority	L2-No. files	L3-Size of files	Calculated value based on Generic Formula (L1*(L2+L3))	Order of Processing
Case 01					
T1	0.4	1	20	8.4	4
T2	0.3	22	30	15.6	3
T3	0.2	65	40	21	2
T4	0.1	200	50	25	1
Case 02					
T1	0.4	2	20	8.8	3
T2	0.3	15	35	15	2
T3	0.2	60	45	21	1
T4	0.1	18	55	7.3	4
Case 03					
T1	0.4	5	30	14	2
T2	0.3	20	50	21	1
T3	0.2	5	45	10	4
T4	0.1	70	60	13	3

a priority value of 25, followed by T3 (priority value: 21), T2 (priority value: 15.6), and T1 (priority value: 8.4), arranged in descending order. This prioritization strategy ensures efficient processing based on a combination of factors, including priority, file quantity, and size associated with each terminal.

Similarly, in Case 02, Terminal T3 takes precedence with the highest calculated value of 21, securing the top processing priority. T2 follows with a value of 15, ranking as the second priority. Terminal T1 holds the third priority with a value of 8.8, while T4 assumes the lowest priority with a value of 7.3.

Moving on to Case 03, Terminal T2 attains the highest calculated value of 21, signifying the foremost order of processing, reflective of its elevated priority and substantial file quantities and sizes. Terminal T1 follows with a calculated value of 14, designating the second processing order. T4 and T3, with calculated values of 13 and 10, respectively, determine their processing order as 3 and 4. The difference in the number of tasks shown in the figures below is due to the hierarchical organization of data processing. Lower levels process individual files or smaller subsets, creating more tasks, while higher levels aggregate data, resulting in fewer, larger tasks. This hierarchy optimizes resource utilization and enhances overall performance.

A. RESULTS FOR CASE 01

The following results are generated for test case 01 when processed using baseline spark and the proposed approach:

The baseline Spark configuration consumed approximately 95.3 MiB, 95.9 MiB, 95.1 MiB, and 94.2 MiB of resources for T1, T2, T3, and T4, respectively (Figure 4). In contrast, the proposed approach demonstrated a marked

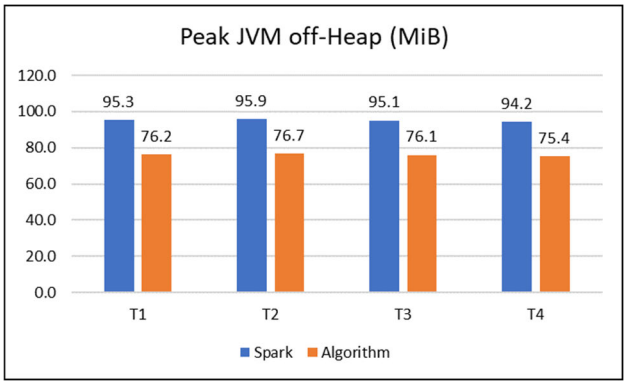

FIGURE 4. Peak JVM off-heap during data processing.

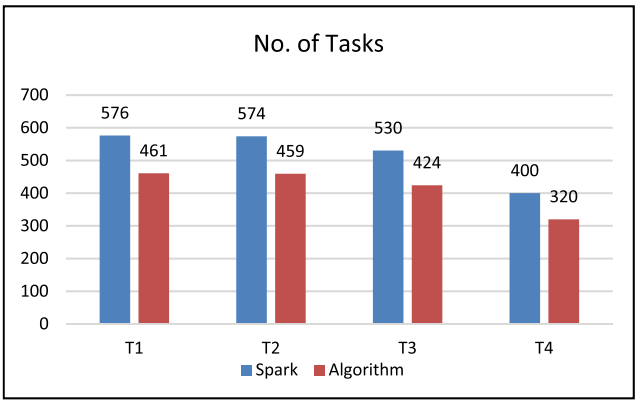

FIGURE 5. No. of tasks generated from data received.

improvement, achieving comparable data processing with significantly lower resource utilization: 76.24 MiB for T1, 76.72 MiB for T2, 76.08 MiB for T3, and 75.36 MiB for T4. This efficiency gain can be attributed to the refined algorithms and streamlined processes introduced in the proposed approach, optimizing memory usage. The observed reduction not only contributes to cost-effectiveness but also holds the potential to enhance overall system performance and scalability.

Figure 5 illustrates a compelling comparative analysis between baseline Spark and the proposed approach in terms of task execution. The baseline Spark required 576, 574, 530, and 400 tasks for processing terminals T1 to T4, respectively, while the proposed approach demonstrated a remarkable improvement, reducing these numbers to 460, 459, 424, and 320 tasks. The substantial decrease in task numbers, especially notable for T4, highlights the effectiveness of the proposed approach in optimizing task execution. This improvement is attributed to the innovative ideology introduced by the proposed approach, showcasing its practicality and efficiency even on a smaller scale. The results underscore not only the adaptability of the proposed approach across various task scales but also its significant impact on enhancing overall system efficiency.

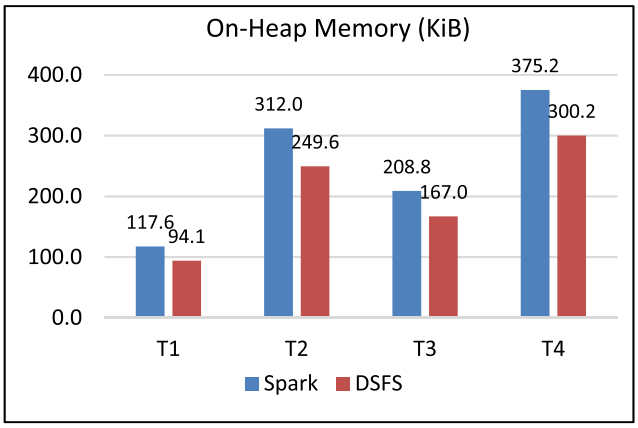

FIGURE 6. On-heap memory usage during data processing.

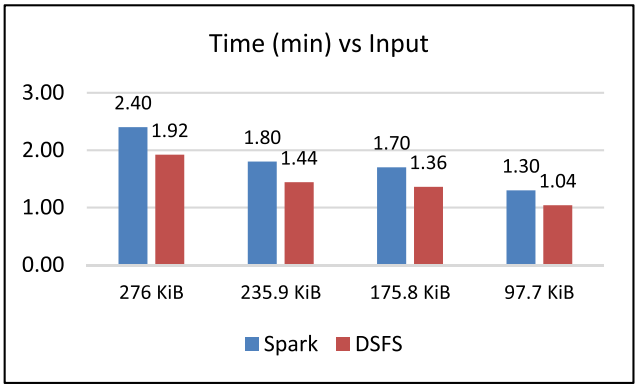

FIGURE 7. Time (min) vs Input processed.

The bar graph shown in Figure 6 depicts on-heap memory usage in kilobytes (KiB) for both Dynamic Smart Flow Scheduler (DSFS) and Spark across four terminals (T1-T4). DSFS consistently exhibits lower memory consumption compared to Spark, with the most significant difference observed at T3, where DSFS utilizes 41.76 KiB less memory (167.04 KiB vs. 198.24 KiB). Specifically, at T1, DSFS consumes 94.08 KiB while Spark uses 117.60 KiB (23.52 KiB difference). This gap widens slightly at T2 (167.04 KiB for DSFS vs. 208.08 KiB for Spark, 62.4 KiB difference) and T4 (300.16 KiB for DSFS vs. 375.20 KiB for Spark, 75.04 KiB difference).

The bar graph in Figure 7 compares the task completion time (seconds) of Dynamic Smart Flow Scheduler (DSFS) and Spark across four test points (T1-T4), likely representing increasing workload complexity. DSFS consistently outperforms Spark in terms of execution speed, with the most significant advantage observed at T1; DSFS finishes execution in 1.92 seconds, compared to Spark's 2.40 seconds (0.48-second difference). This gap changes slightly at T2, where DSFS takes 1.44 seconds to complete the task, while Spark takes 1.80 seconds (0.36 seconds difference). The trend continues at T3, where DSFS executes in 1.36 seconds, maintaining a 0.34-second lead over Spark's 1.70 seconds.

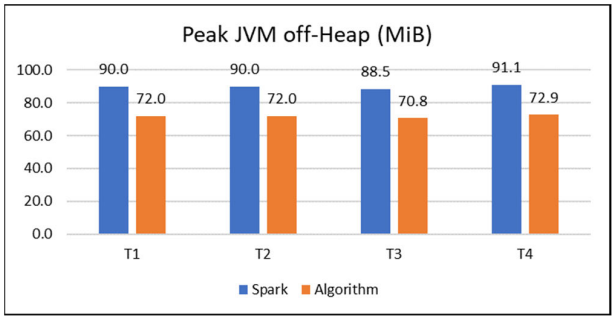

FIGURE 8. Peak JVM off-heap during data processing.

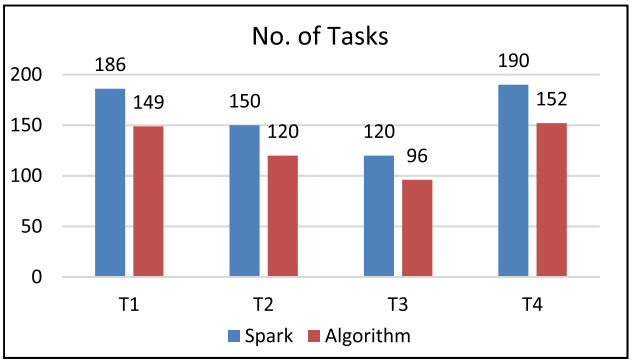

FIGURE 9. No. of tasks generated from data received.

Moreover, at T4, where DSFS completes the task in 1.04 seconds, a 0.26-second improvement over Spark's 1.30 seconds.

B. RESULTS FOR CASE 02

In Figure 8, a comparison of Spark configurations reveals significant improvements in resource efficiency with the proposed approach. The initial Spark configuration, as depicted in the results, required approximately 90 MiB for T1, T2, 88.5 MiB for T3, and 91.1 MiB for T4. In contrast, the proposed approach not only maintained equivalent data processing capabilities but also exhibited a considerable reduction in resource consumption. The streamlined configuration required only 72 MiB for T1, 72 MiB for T2, 70.8 MiB for T3, and 72.88 MiB for T4. This reduction in resource utilization indicates the proposed approach's ability to optimize resource allocation, resulting in more efficient processing across all terminals. The observed decrease in resource consumption underscores the potential benefits of the proposed methodology in enhancing the overall performance and resource utilization efficiency of the Spark framework for the specified tasks.

In Figure 9, the baseline Spark execution deployed a total of 186 tasks for processing T1, 150 tasks for T2, 120 tasks for T3, and 190 tasks for T4. The observed reduction in task numbers with the proposed approach exemplifies its efficacy in optimizing task allocation and execution. Specifically, the proposed approach demonstrated a significant decrease to 149 tasks for T1, 120 tasks for T2, 96 tasks for T3, and

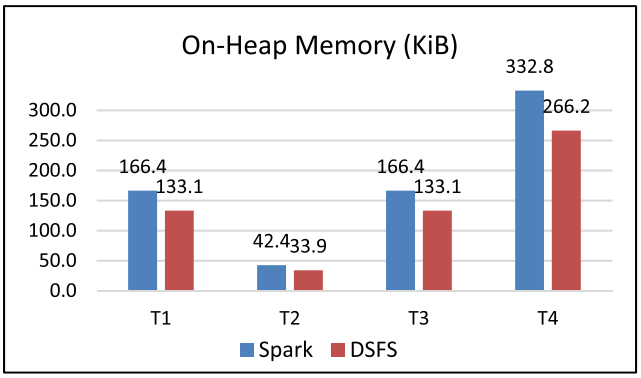

FIGURE 10. On-heap memory usage during task processing.

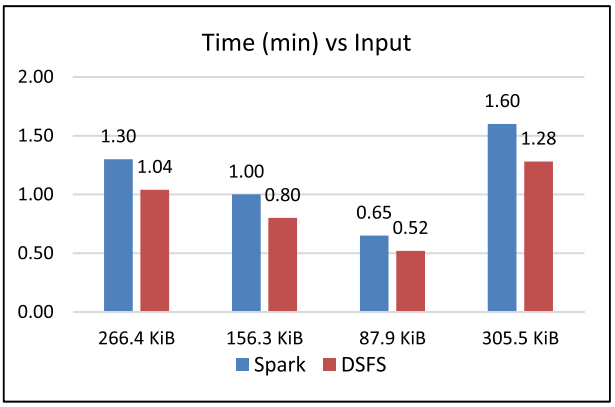

FIGURE 11. Time (min) vs Input processed.

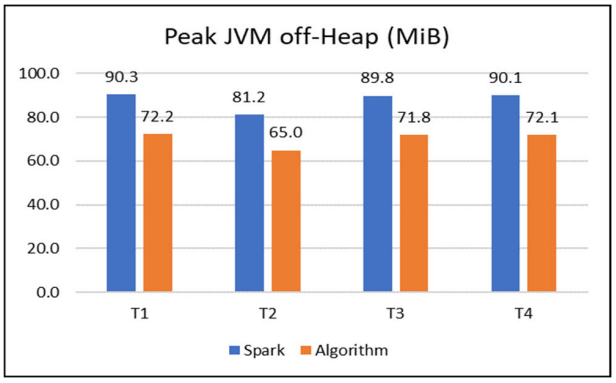

FIGURE 12. Peak JVM Off-Heap during data processing.

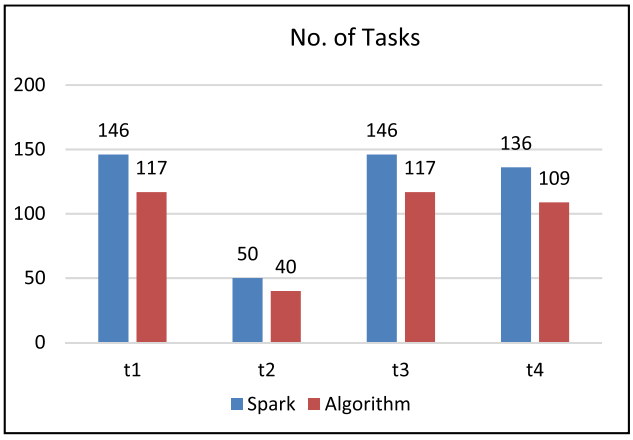

FIGURE 13. No. of tasks generated from data received.

152 tasks for T4. This reduction can be attributed to the improved algorithm or methodology implemented, resulting in a more streamlined and efficient task distribution across the terminals. The noteworthy decrease in task numbers not only indicates enhanced performance but also suggests a potential reduction in computational overhead and resource utilization.

The bar graph in Figure 10 depicts the on-heap memory usage in kilobytes (KiB) for both Spark and DSFS across four test points (T1-T4), likely representing increasing workload complexity. DSFS consistently exhibits lower on-heap memory consumption compared to Spark across all terminals. The nominal difference occurs at T1 and T3, where DSFS utilizes 33.2 KiB less memory than Spark (133.1 KiB vs. 166.4 KiB). Notably, at T2, DSFS consumes 8.5 KiB less memory (33.9 KiB vs. 42.4 KiB). The trend continues at T4, where DSFS uses 66.56 KiB less memory than Spark (266.24 KiB vs. 332.80 KiB).

The bar graph in Figure 11 depicts the execution time in minutes for both Spark and DSFS across four terminals (T1-T4). DSFS consistently outperforms Spark in terms of execution speed across all terminals. The most significant advantage is observed at T4, where DSFS completes the task in 1.28 minutes, a 0.32-minute improvement over Spark's 1.60 minutes. Similarly, at T1, DSFS finishes execution in 1.04 minutes, compared to Spark's 1.30 minutes (0.26-minute difference). This gap narrows slightly at T2, where DSFS takes 0.80 minutes to complete the task, while Spark takes 1.00 minutes (0.20-minute difference). The trend continues at T3, where DSFS executes in 0.52 minutes, maintaining a 0.13-minute lead over Spark's 0.65 minutes completion time.

C. RESULTS FOR CASE 03

Illustrated in Figure 12, the initial Spark configuration consumed approximately 90.3 MiB, 81.2 MiB, 89.8 MiB, and 90.1 MiB of resources for terminals T1, T2, T3, and T4, respectively. In contrast, the proposed approach not only achieved comparable data processing but also demonstrated a notable reduction in resource utilization. This reduction is evident in the decreased resource consumption values of 72.24 MiB for T1, 64.96 MiB for T2, 71.84 MiB for T3, and 72.08 MiB for T4. The improved efficiency in resource allocation can be attributed to optimizations employed in the proposed approach. This not only underscores the effectiveness of the new methodology but also highlights its potential for resource optimization in Spark-based data processing tasks.

In Figure 13, the baseline Spark execution is compared to the proposed approach, revealing a noteworthy optimization in task allocation. The baseline execution initially required

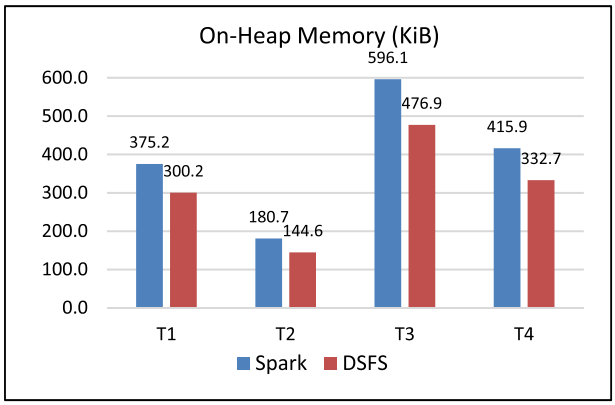

FIGURE 14. On-heap memory usage during data processing.

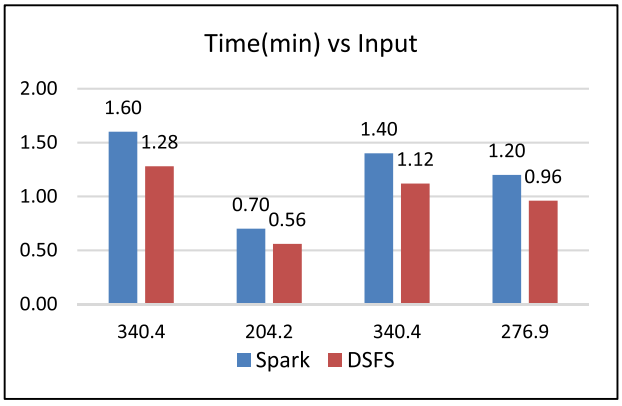

FIGURE 15. Time (min) vs Input processed.

146 tasks for processing T1, 50 for T2, 146 for T3, and 136 for T4. In contrast, the proposed approach demonstrates a significant reduction in task numbers, allocating only 117 tasks for T1, 40 for T2, 117 for T3, and 109 for T4. This reduction is indicative of the efficiency gained through the proposed approach, resulting in a streamlined processing workflow. The decrease in task count suggests improved resource utilization and enhanced parallelism, contributing to overall performance enhancement.

The bar graph in Figure 14 depicts the on-heap memory usage in kilobytes (KiB) for both Spark and DSFS across four terminals (T1-T4). DSFS consistently demonstrates lower on-heap memory consumption compared to Spark across all terminals. The most prominent difference occurs at T3, where DSFS utilizes 119.22 KiB less memory than Spark (476.88 KiB vs. 596.10 KiB). The trend continues at T4, where DSFS uses 83.18 KiB less memory than Spark (332.72 KiB vs. 415.90 KiB). At T1, the difference of 75.04 KiB (DSFS 300.16 KiB vs Spark 375.20 KiB) can be seen. Similarly, at T2, DSFS consumes 36.14 KiB less memory (144.56 KiB vs. 180.70 KiB).

The bar graph shown in Figure 15 depicts the execution time in minutes for both Spark and DSFS across the input of four terminals (T1-T4). DSFS outperforms Spark in terms of execution speed across all terminals. At T1, DSFS

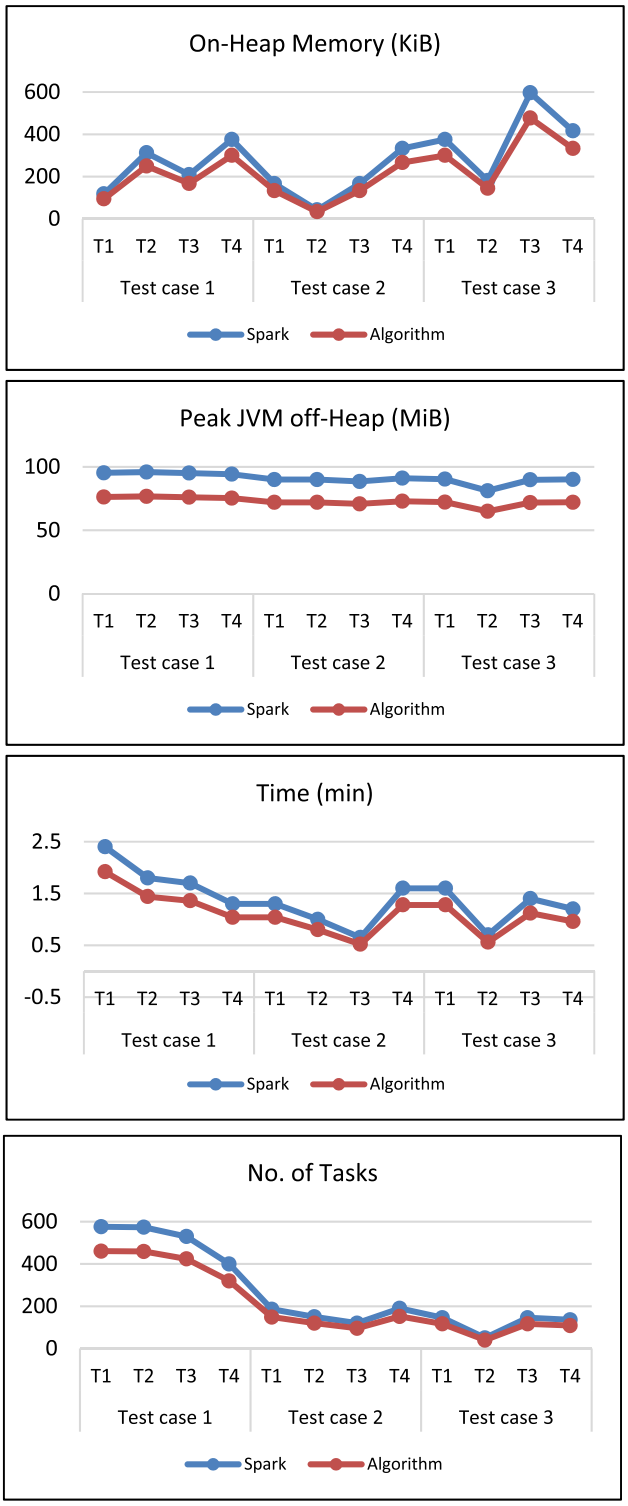

FIGURE 16. Spark vs DSFS algorithm comparison.

completes the task in 1.28 minutes, a 0.32-minute improvement over Spark's 1.60 minutes. Notably, at T2 and T3, DSFS finishes execution in 0.56 minutes and 1.12 minutes, respectively, maintaining a lead over Spark's 0.70 minutes and 1.40 minutes. Whereas, at T4, Spark finishes execution in 1.20 minutes compared to DSFS, finishing in 0.96 min.

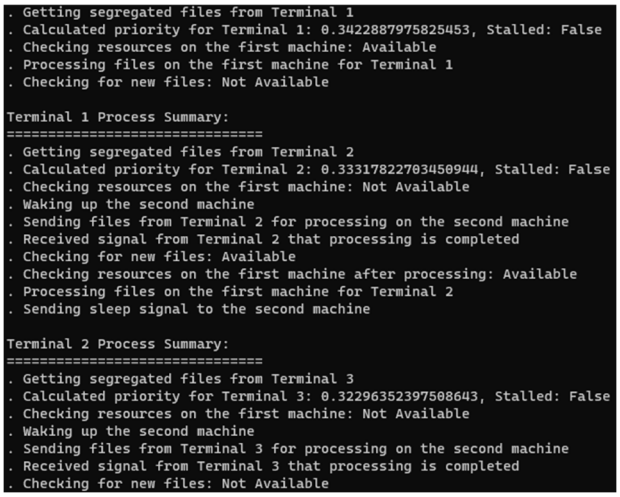

FIGURE 17. Energy efficiency.

In Figure 16, the line graphs depict a summary of how the proposed algorithm is capable of enhancing Spark's workings in terms of generating tasks, memory and time consumption.

D. ENERGY EFFICIENCY RESULTS

In Figure 17, the proposed method incorporates a dynamic system that intelligently manages the activation and deactivation of the second machine. When ample resources are present on the first machine to handle terminal processing, the second machine is efficiently set to sleep. Conversely, if the first machine surpasses a utilization threshold of 70% (denoted as "Not Available"), the script signals the second machine to wait and proceeds to process the files on it. This strategy avoids the continuous operation of the second machine, contributing to energy efficiency. Once the processing concludes and sufficient resources on the first machine are detected for upcoming data, the script signals the second machine to enter a sleep state, further optimizing energy consumption.

VI. CONCLUSION AND FUTURE WORK

The proposed approach consistently demonstrated substantial improvements across three cases, as indicated by statistical values. In Result 01, resource consumption was reduced by 19.9%, from an average of 94.1 MiB to 75.6 MiB, accompanied by a 24.3% reduction in the number of tasks from an average of 497.5 to 376.5. Similarly, result 02 showed a 20.3% reduction in resource consumption, from an average of 90.4 MiB to 72.08 MiB, coupled with an 18.6% decrease in tasks from an average of 161 to 131. In Result 03, a consistent pattern emerged with a 17.5% reduction in resource consumption, from an average of 87.6 MiB to 72.26 MiB, and a 5.3% decrease in tasks from an average of 132 to 125. These findings underscore the proposed approach's consistent and significant improvement in resource efficiency and task optimization. The observed reductions in off-heap consumption and task execution in Apache Spark contribute to enhanced system performance, resource utilization, and scalability. Notable benefits include optimized garbage collection, increased available heap for application data, and lower operational costs. Additionally, the reduction in task consumption aligns with the overarching goal of achieving optimal performance in Spark applications, fostering a more efficient and responsive distributed data-processing environment. The difference in the number of tasks is due to the hierarchical organization of the data processing tasks. While the total number of files to be processed remains constant, the hierarchical structure introduces variations in task complexity and granularity. Tasks at lower levels of the hierarchy involve processing individual files or smaller subsets of data, resulting in a larger number of tasks. Conversely, tasks at higher levels of the hierarchy aggregate and process data from multiple sources or at a coarser granularity, resulting in fewer tasks, each handling a larger volume of data. This hierarchical decomposition allows for efficient organization and management of the data processing pipeline, optimizing resource utilization and enhancing overall performance. This approach not only enhances reliability and effectiveness, particularly in large-scale data processing scenarios involving Apache Spark, but also aligns with sustainability objectives by lowering resource utilization and promoting energy savings. In conclusion, implementing a data scheduling system in a smart city setting has demonstrated diverse positive effects on city management. Leveraging Spark's distributed computing capabilities has allowed the system to scale efficiently in response to the growing volume of data, ensuring adaptability as the smart city expands. The strategic allocation of computational resources to high-priority data streams has enhanced overall resource management within the city's data infrastructure, thereby optimizing efficiency. Real-time access to critical data has emerged as a fundamental benefit that enables city authorities to make informed decisions promptly. The Python and Spark-based data scheduling system, with its focus on prioritizing data streams, represents a significant advancement in smart city management. It facilitates prompt and efficient handling of critical data and fosters a highly responsive city infrastructure. Furthermore, the proposed approach yielded substantial reductions in off-heap consumption and the total number of tasks executed in Apache Spark. These reductions have translated into enhanced system performance, resource utilization, and scalability, with notable benefits such as optimized garbage collection, increased available heap for application data, and lower operational costs. This signifies a positive impact on the reliability and effectiveness of the systems, particularly in large-scale data processing scenarios involving Apache Spark. Additionally, the reduction in task consumption contributes to a more efficient and responsive distributed data-processing environment, aligning with the overarching goal of achieving optimal performance in Spark applications. An ancillary advantage lies in lowered resource utilization, fostering energy savings, and aligning with sustainability objectives. In summary, the multifaceted improvements brought about by the data scheduling system underscore its

pivotal role in shaping a more effective, responsive, and sustainable smart-city management framework. These findings provide valuable insights into the broader discourse on enhancing data-processing efficiency and resource utilization in dynamic urban environments. Future recommendations emphasize enhancing scalability and resource utilization by leveraging task parallelism and dynamic load balancing for optimal performance under heavy workloads. Adaptive resource management, including real-time dynamic node allocation, will optimize resource use. The energy-efficient design should continue to scale effectively, reducing energy consumption while maintaining performance. Strengthening fault tolerance mechanisms will ensure system stability despite node failures. Experimentation results validating scalability with consistent performance metrics suggest future strategies such as centralized monitoring, adaptive scaling, distributed task coordination, and consistency management. Further research will focus on enhancing scalability and addressing challenges identified during initial testing, ensuring continued improvements in system performance, resource utilization, and scalability in large-scale data processing scenarios.

ACKNOWLEDGMENT

The authors would like to acknowledge and thank Ajman University, United Arab Emirates, and Universiti Sains Malaysia for their support.

REFERENCES

[1] J. M. Shapiro, "Smart cities: Quality of life, productivity, and the growth effects of human capital," *Rev. Econ. Statist.*, vol. 88, no. 2, pp. 324–335, May 2006, doi: 10.1162/rest.88.2.324.

[2] A. Luberg, T. Tammet, and P. Järv, "Smart city: A rule-based tourist recommendation system," in *Information and Communication Technologies in Tourism*. Vienna: Springer, 2011, pp. 51–62.

[3] K. Su, J. Li, and H. Fu, "Smart city and the applications," in *Proc. Int. Conf. Electron., Commun. Control (ICECC)*, 2011, pp. 1028–1031, doi: 10.1109/ICECC.2011.6066743.

[4] K. C. Seto, B. Güneralp, and L. R. Hutyra, "Global forecasts of urban expansion to 2030 and direct impacts on biodiversity and carbon pools," *Proc. Nat. Acad. Sci. USA*, vol. 109, no. 40, pp. 16083–16088, Oct. 2012, doi: 10.1073/pnas.1211658109.

[5] S. E. Bibri and J. Krogstie, "Smart sustainable cities of the future: An extensive interdisciplinary literature review," *Sustain. Cities Soc.*, vol. 31, pp. 183–212, May 2017.

[6] C. Harrison, B. Eckman, R. Hamilton, P. Hartswick, J. Kalagnanam, J. Paraszczak, and P. Williams, "Foundations for smarter cities," *IBM J. Res. Develop.*, vol. 54, no. 4, pp. 1–16, Jul. 2010, doi: 10.1147/JRD.2010.2048257.

[7] M. Herrera, E. Abraham, and I. Stoianov, "A graph-theoretic framework for assessing the resilience of sectorised water distribution networks," *Water Resour. Manage.*, vol. 30, no. 5, pp. 1685–1699, Mar. 2016.

[8] M. Herrera, L. Torgo, J. Izquierdo, and R. Pérez-García, "Predictive models for forecasting hourly urban water demand," *J. Hydrol.*, vol. 387, nos. 1–2, pp. 141–150, Jun. 2010.

[9] N. M. Mirza, R. Bader, A. Ali, and M. K. Ishak, "Leveraging the Internet of Things aware healthcare monitoring system for better living standards," in *Proc. 7th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Dec. 2022, pp. 1–5, doi: 10.1109/FMEC57183.2022.10062818.

[10] I. Santosa, S. H. Supangkat, and A. A. Arman, "Value of smart city services in improving the quality of life: A literature review," in *Proc. 10th Int. Conf. ICT Smart Soc. (ICISS)*, Bandung, Indonesia, 2023, pp. 1–6, doi: 10.1109/iciss59129.2023.10291571.

[11] H. K. Hettikankanama, "Integrating smart transportation system for a proposed smart city: A mapping study," in *Proc. Int. Res. Conf. Smart Comput. Syst. Eng. (SCSE)*, 2019, pp. 196–203.

[12] M. Paturi, S. Puvvada, B. S. Ponnuru, M. Simhadri, B. S. Egala, and A. K. Pradhan, "Smart solid waste management system using blockchain and IoT for smart cities," in *Proc. IEEE Int. Symp. Smart Electron. Syst. (iSES)*, Dec. 2021, pp. 456–459, doi: 10.1109/iSES52644.2021.00107.

[13] J. Yang, Y. Kwon, and D. Kim, "Regional smart city development focus: The south Korean national strategic smart city program," *IEEE Access*, vol. 9, pp. 7193–7210, 2021, doi: 10.1109/ACCESS.2020.3047139.

[14] N. Komninos, H. Schaffers, and M. Pallot, "Developing a policy roadmap for smart cities and the future internet," in *Proc. EChallenges E-2011 Conf.*, 2011, pp. 286–306.

[15] C. B. Rubin, A. Jones, and A. Kovacich, *Emergency Management: The American Experience*. Boca Raton, FL, USA: CRC Press, 2012, pp. 1900–2010.

[16] A. Golyan, S. Panchal, D. Vaghasiya, and H. Parekh, "Data ethics and privacy," in *Recent Trends and Future Direction for Data Analytics*. Hershey, PA, USA: IGI Global, 2024, pp. 259–268.

[17] A. Kumari, A. Golyan, R. Shah, and N. Raval, "Introduction to data analytics," in *Recent Trends and Future Direction for Data Analytics*. Hershey, PA, USA: IGI Global, 2024, pp. 1–14.

[18] A. Kumari, M. M. Patel, A. Shukla, S. Tanwar, N. Kumar, and J. J. P. C. Rodrigues, "ArMor: A data analytics scheme to identify malicious behaviors on blockchain-based smart grid system," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Jul. 2020, pp. 1–6.

[19] A. Kumari, A. Shukla, R. Gupta, S. Tanwar, S. Tyagi, and N. Kumar, "ET-DeaL: A P2P smart contract-based secure energy trading scheme for smart grid systems," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 1051–1056.

[20] A. Kumari and S. Tanwar, "RAKSHAK: Resilient and scalable demand response management scheme for smart grid systems," in *Proc. 11th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2021, pp. 309–314.

[21] A. Kumari, U. C. Sukharamwala, S. Tanwar, M. S. Raboaca, F. Alqahtani, A. Tolba, R. Sharma, I. Aschilean, and T. C. Mihaltan, "Blockchain-based peer-to-peer transactive energy management scheme for smart grid system," *Sensors*, vol. 22, no. 13, p. 4826, Jun. 2022.

[22] A. Kumari and S. Tanwar, "A data analytics scheme for security-aware demand response management in smart grid system," in *Proc. IEEE 7th Uttar Pradesh Sect. Int. Conf. Electr., Electron. Comput. Eng. (UPCON)*, Nov. 2020, pp. 1–6.

[23] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. OSDI*, 2008, pp. 29–42.

[24] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment," in *Proc. 10th IEEE Int. Conf. Comput. Inf. Technol.*, Jun. 2010, pp. 2736–2743, doi: 10.1109/CIT.2010.458.

[25] L. Lei, T. Wo, and C. Hu, "CREST: Towards fast speculation of straggler tasks in MapReduce," in *Proc. IEEE 8th Int. Conf. e-Business Eng.*, Oct. 2011, pp. 311–316, doi: 10.1109/ICEBE.2011.37.

[26] M. Hammoud and M. F. Sakr, "Locality-aware reduce task scheduling for MapReduce," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2011, pp. 570–576, doi: 10.1109/CLOUDCOM.2011.87.

[27] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Maestro: Replica-aware map scheduling for MapReduce," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (ccgrid)*, May 2012, pp. 435–442, doi: 10.1109/CCGRID.2012.122.

[28] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new MapReduce scheduling technique," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2011, pp. 40–47, doi: 10.1109/CLOUDCOM.2011.16.

[29] P. Nguyen, T. Simon, M. Halem, D. Chapman, and Q. Le, "A hybrid scheduling algorithm for data intensive workloads in a MapReduce environment," in *Proc. IEEE 5th Int. Conf. Utility Cloud Comput.*, Nov. 2012, pp. 161–167, doi: 10.1109/UCC.2012.32.

[30] A. Gounaris, G. Kougka, R. Tous, C. T. Montes, and J. Torres, "Dynamic configuration of partitioning in spark applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 1891–1904, Jul. 2017.

[31] A. K. Javanmardi, S. H. Yaghoubyan, K. BagheriFard, S. Nejatian, and H. Parvin, "An architecture for scheduling with the capability of minimum share to heterogeneous Hadoop systems," *J. Supercomput.*, vol. 77, no. 6, pp. 5289–5318, Jun. 2021.

[32] P. Zhang, Y. Li, H. Lin, J. Wang, and C. Zhang, "A periodic task-oriented scheduling architecture in cloud computing," in *Proc. IEEE Intl Conf Parallel Distrib. Process. Appl., Ubiquitous Comput. Commun., Big Data Cloud Comput., Social Comput. Netw., Sustain. Comput. Commun. (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Jul. 2018, pp. 788–794.

[33] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and K. Li, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Apr. 2017.

[34] V. P. Verma, N. S. Naik, and S. Kumar, "Reinforcement learning based scheduling for spark jobs in cloud environment," in *Proc. IEEE 9th Uttar Pradesh Sect. Int. Conf. Electr., Electron. Comput. Eng. (UPCON)*, Dec. 2022, pp. 1–6, doi: 10.1109/UPCON56432.2022.9986440.

[35] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 71–91, Jan. 2022, doi: 10.1109/TKDE.2020.2975652.

[36] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "MapTask scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 190–203, Feb. 2016, doi: 10.1109/TNET.2014.2362745.

[37] N. S. Naik, A. Negi, and R. Anitha, "A data locality based scheduler to enhance MapReduce performance in heterogeneous environments," *Future Gener. Comput. Syst.*, vol. 90, pp. 423–434, Jan. 2019, doi: 10.1016/j.future.2018.07.043.

[38] O. Selvitopi, G. V. Demirci, A. Turk, and C. Aykanat, "Locality-aware and load-balanced static task scheduling for MapReduce," *Future Gener. Comput. Syst.*, vol. 90, pp. 49–61, Jan. 2019, doi: 10.1016/j.future.2018.06.035.

[39] D. Choi, M. Jeon, N. Kim, and B.-D. Lee, "An enhanced data-locality-aware task scheduling algorithm for Hadoop applications," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3346–3357, Dec. 2018, doi: 10.1109/JSYST.2017.2764481.

[40] O. Beaumont, T. Lambert, L. Marchal, and B. Thomas, "Data-locality aware dynamic schedulers for independent tasks with replicated inputs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 1206–1213, doi: 10.1109/IPDPSW.2018.00187.

[41] G. Wang, J. Xu, and B. He, "A novel method for tuning configuration parameters of spark based on machine learning," in *Proc. IEEE 18th Int. Conf. High Perform. Comput. Communications; IEEE 14th Int. Conf. Smart City; IEEE 2nd Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Dec. 2016, pp. 586–593.

[42] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna, "Stage aware performance modeling of DAG based in memory analytic platforms," in *Proc. IEEE 9th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2016, pp. 188–195.

[43] K. Wang and M. M. H. Khan, "Performance prediction for apache spark platform," in *Proc. IEEE IEEE 17th Int. Conf. High Perform. Comput. Commun. 7th Int. Symp. Cyberspace Saf. Secur., IEEE 12th Int. Conf. Embedded Softw. Syst.*, Aug. 2015, pp. 166–173.

[44] M. T. Islam, S. Karunasekera, and R. Buyya, "DSpark: Deadline-based resource allocation for big data applications in apache spark," in *Proc. IEEE 13th Int. Conf. e-Science (e-Science)*, Oct. 2017, pp. 89–98.

[45] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, Apr. 2014.

[46] S. A. Jyothi, "Morpheus: Towards automated SLOs for enterprise clusters," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 117–134.

[47] S. Dimopoulos, C. Krintz, and R. Wolski, "Justice: A deadline-aware, fair-share resource allocator for implementing multi-analytics," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2017, pp. 233–244.

[48] J. S. Damji, B. Wenig, T. Das, and D. Lee, *Learning Spark*. Sebastopol, CA, USA: O'Reilly Media, 2020.

[49] D. Cheng, Y. Chen, X. Zhou, D. Gmach, and D. Milojicic, "Adaptive scheduling of parallel jobs in spark streaming," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[50] T. Ajila and S. Majumdar, "Data driven priority scheduling on spark based stream processing," in *Proc. IEEE/ACM 5th Int. Conf. Big Data Comput. Appl. Technol. (BDCAT)*, Dec. 2018, pp. 208–210.

[51] P. Garefalakis, K. Karanasos, and P. Pietzuch, "Neptune: Scheduling suspendable tasks for unified stream/batch applications," in *Proc. ACM Symp. Cloud Comput.*, Nov. 2019, pp. 233–245.

[52] M. Natarajan and S. Subramanian, "A cross-layer design: Energy efficient multilevel dynamic feedback scheduling in wireless sensor networks using deadline aware active time quantum for environmental monitoring," *Int. J. Electron.*, vol. 106, no. 1, pp. 87–108, Jan. 2019.

NADA M. MIRZA received the bachelor's and master's degrees in mechatronics engineering from the College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. She is currently pursuing the Ph.D. degree with Universiti Sains Malaysia (USM). She also holds the position of an Instructor with the College of Engineering, United Arab Emirates University (UAEU), United Arab Emirates. Since 2014, she has been actively engaged in academic activities pertaining to control and electronics engineering in Al Ain, United Arab Emirates. Her postgraduate research primarily concentrated on the utilization of artificial intelligence, robotics, and wireless monitoring systems for renewable energy applications. With a background in academia in Pakistan, her research primarily revolved around autonomous wireless intelligent robotic systems. Her research interests include robotics, artificial intelligence, the Internet of Things (IoT), and control systems.

ADNAN ALI received the B.Eng. degree in software engineering from Al Ain University (AAU), in 2020, with a focus on virtual reality educational applications, and the M.S. degree in engineering from Universiti Sains Malaysia (USM). He is currently a Web Developer with the University of Khorfakkan, United Arab Emirates. His research ended in the publication of a paper based on his final year project. His research interests include web development, big data, the IoT, embedded systems, simulation, and virtual reality.

NURA SHIFA MUSA received the bachelor's degree in computer engineering from Qatar University (QU), Qatar, and the master's degree in information security from the College of Information Technology, United Arab Emirates University (UAEU), United Arab Emirates. She is currently a Senior Laboratory Supervisor with the College of Engineering, Al Ain University (AAU), United Arab Emirates. Demonstrating a profound dedication to advancing cyber security measures, her research interests include developing innovative solutions to enhance digital security, investigating cyber threats, exploring cloud computing technology, and conducting digital forensic investigations. She received awards and honors.

MOHAMAD KHAIRI ISHAK (Member, IEEE) received the B.Eng. degree in electrical and electronics engineering from International Islamic University Malaysia (IIUM), Malaysia, the M.Sc. degree in embedded systems from the University of Essex, U.K., and the Ph.D. degree from the University of Bristol, U.K. He is a Registered Graduate Engineer with the Board of Engineers Malaysia (BEM). He is currently an Associate Professor Lecturer in computer engineering with the Department of Electrical and Computer Engineering, Ajman University, United Arab Emirates. His background includes outstanding teaching experience at Universiti Sains Malaysia and Ajman University, United Arab Emirates, instructing students to stimulate engineering information interest and retention while invigorating classes through the use of new technologies and models. Emphasis is given to the development of theoretical and practical methods that can be practically validated. Recently, significant research efforts have been directed toward important industrial issues of embedded networked control systems with AI and IoT. His research interests include embedded systems, artificial intelligence, real-time control communications, and the Internet of Things (IoT).