electronics

MDPI

Article

Adaptive Multi-Criteria Selection for Efficient Resource Allocation in Frugal Heterogeneous Hadoop Clusters

Basit Qureshi

Department of Computer Science, Prince Sultan University, Riyadh 11586, Saudi Arabia; qureshi@psu.edu.sa

Abstract: Efficient resource allocation is crucial in clusters with frugal Single-Board Computers (SBCs) possessing limited computational resources. These clusters are increasingly being deployed in edge computing environments in resource-constrained settings where energy efficiency and cost-effectiveness are paramount. A major challenge in Hadoop scheduling is load balancing, as frugal nodes within the cluster can become overwhelmed, resulting in degraded performance and frequent occurrences of out-of-memory errors, ultimately leading to job failures. In this study, we introduce an Adaptive Multi-criteria Selection for Efficient Resource Allocation (AMS-ERA) in Frugal Heterogeneous Hadoop Clusters. Our criterion considers CPU, memory, and disk requirements for jobs and aligns the requirements with available resources in the cluster for optimal resource allocation. To validate our approach, we deploy a heterogeneous SBC-based cluster consisting of 11 SBC nodes and conduct several experiments to evaluate the performance using Hadoop wordcount and terasort benchmark for various workload settings. The results are compared to the Hadoop-Fair, FOG, and IDaPS scheduling strategies. Our results demonstrate a significant improvement in performance with the proposed AMS-ERA, reducing execution time by 27.2%, 17.4%, and 7.6%, respectively, using terasort and wordcount benchmarks.

Keywords: frugal Hadoop clusters; dynamic analytical hierarchy process; locality-aware data placement; single-board computers

Citation: Qureshi, B. Adaptive Multi-Criteria Selection for Efficient Resource Allocation in Frugal Heterogeneous Hadoop Clusters. *Electronics* **2024**, *13*, 1836. https://doi.org/10.3390/electronics13101836

Received: 15 April 2024
Revised: 3 May 2024
Accepted: 7 May 2024
Published: 9 May 2024

1. Introduction

Frugal computing refers to the practice of designing, building, and deploying computing systems with a focus on cost-effectiveness, resource efficiency, and sustainability. The term "frugal" implies simplicity, economy, and minimalism, where the goal is to meet computing needs with the least number of resources, both in terms of hardware and energy [1]. Frugal clusters are an innovative solution that intersects sustainability and digital transformation [2]. By leveraging energy-efficient hardware components like Single-Board Computers (SBCs) [3], these clusters reduce energy consumption, aligning with sustainability goals and minimizing environmental impact [4], thus aligning with broader sustainability goals. Moreover, their cost-efficient nature makes them accessible to organizations with limited budgets, democratizing access to big data processing capabilities and fostering inclusivity in digital transformation initiatives [1]. Frugal clusters prioritize resource optimization through adaptive resource allocation and workload-aware scheduling, ensuring efficient resource utilization and maximizing performance.

Hadoop, an open-source framework, facilitates the distributed processing of large datasets across computer clusters using simple programming models. A key distinction of Hadoop is its integration of both storage and computation within the same framework. Unlike traditional methods, Hadoop allows for the flexible movement of computation, primarily MapReduce jobs, to the location of the data, managed by a Hadoop Distributed File System (HDFS). Consequently, efficient data placement within compute nodes becomes essential for effective big data processing [5]. Hadoop's default approach to data locality relies heavily on the physical proximity of data to computation nodes, which may not

always guarantee optimal performance. However, this feature overlooks other important factors such as network congestion, node availability, and load balancing, which can significantly impact data access latency and overall job execution time [6]. Additionally, Hadoop's default data locality mechanism does not take into account the heterogeneity of cluster nodes, including variations in processing power, memory capacity, and disk I/O capabilities [7,8]. As a result, tasks may be assigned to nodes that are ill suited for processing them efficiently, leading to resource contention and reduced performance. Furthermore, the default data locality mechanism may not dynamically adapt to changing cluster conditions or workload patterns, resulting in suboptimal resource utilization and wasted computational resources.

In recent times, researchers have addressed the optimal resource allocation for scheduling issues in heterogeneous Hadoop clusters. Z. Guo and G. Fox. [9] introduced techniques like speculative execution to mitigate the impact of slow nodes, thereby optimizing resource utilization and job completion times. The study emphasizes the importance of efficient resource management and scheduling algorithms to improve overall performance in environments with varying computational capabilities. In [10], Bae notes that in heterogeneous environments, Hadoop's subpar performance was observed due to equal block allocation across nodes in the cluster. They proposed a new data placement scheme aimed at improving Hadoop's data locality while minimizing replicated data by selecting and replicating only blocks with the highest likelihood of remote access. In [11], Bawankule K. presents a historical data-based data placement (HDBDP) policy to balance the workload among heterogeneous nodes. Their approach is based on the node's computing capabilities to improve the map tasks' data locality and to reduce the job turnaround time in the heterogeneous Hadoop environment. Resource- and Network-aware Data Placement Algorithm (RENDA) for resource- and network-aware data placement in Hadoop is presented in [12]. The RENDA reduces the time of the data distribution and data processing stages by estimating the heterogeneous performance of the nodes on a real-time basis. It carefully allocates data blocks in several installments to participating nodes in the cluster.

The researchers in [13] discuss the development of a novel job scheduler, CLQLMRS, using reinforcement learning to improve data and cache locality in MapReduce job scheduling, highlighting the importance of reducing job execution time for enhancing Hadoop performance. In [14], the authors propose a DQ-DCWS algorithm to balance data locality and delays in Hadoop while considering five Quality of Service factors. The DQ-DCWS is based on dynamic programming in calculating the length of the edge in the DAG and scheduling tasks along the optimal path. In [15], Postoaca et al. presented a deadline-aware fog Scheduler (FOG) for cloud edge applications. The job queue is ordered for context based on deadlines. The nodes in the cluster are ordered using a similarity index. The highest-ordered jobs are sorted and assigned to the appropriate clusters. The authors in [16] propose an Improved Data Placement Strategy (IDaPS) based on intra-dependency among data blocks to enhance performance and reduce data transfer overheads. The proposed IDaPS uses the Markov clustering algorithm to characterize MapReduce task execution based on intra-dependency and task execution frequency.

This paper addresses the challenge of efficient resource allocation in frugal Hadoop clusters. We propose an Adaptive Multi-criteria Selection for Efficient Resource Allocation (AMS-ERA) in Frugal Heterogeneous Hadoop Clusters. Our criterion considers CPU, memory, and disk requirements for jobs and aligns the requirements with available resources in the cluster for optimal resource allocation. Resources available in the cluster are profiled and ranked based on similarity and proximity using the K-means clustering method. A dynamic Analytical Hierarchy Process (dAHP) determines the optimal placement of a job using a score vector to determine the best possible node for a job. The process involves refining the AHP model's accuracy by integrating historical information obtained through Hadoop APIs to assign weights to jobs based on their resource requirements. Finally, the jobs are assigned to the most appropriate nodes, ensuring load balancing in the heterogeneous cluster. These strategies aim to optimize data layout in Hadoop by

maximizing parallelism while accommodating the resource constraints of frugal SBC nodes in the Hadoop cluster. To validate the proposed AMS-ERA, we deploy a heterogeneous SBC-based cluster consisting of 11 physical nodes and execute the Hadoop benchmark tests to analyze the performance of the proposed technique against Hadoop-Fair, FOG [15], and IDaPS [16] scheduling strategies. The results showcase a notable enhancement in performance with our proposed approach. Our results demonstrate a significant improvement in performance with the proposed AMS-ERA, reducing execution time by 27.2%, 17.4%, and 7.6%, respectively, using terasort and wordcount benchmarks. The contributions of this work are threefold:

- We introduce the AMS-ERA approach to optimize resource allocation in frugal Hadoop clusters with Single-Board Computers (SBCs). By considering CPU, memory, and disk requirements for jobs, and aligning these with available resources, AMS-ERA enhances resource allocation to improve performance and efficiency.
- The proposed method involves profiling available resources in the cluster using K-means clustering and dynamically placing jobs based on a refined Analytical Hierarchy Process (AHP). This dynamic placement ensures optimal resource utilization and load balancing in heterogeneous clusters.
- We construct a heterogeneous 11-node Hadoop cluster using popular SBC devices to validate our approach. The work demonstrates that AMS-ERA achieves significant performance improvements compared to other scheduling strategies like Hadoop-Fair, FOG, and IDaPS using various IO-intensive and CPU-intensive Hadoop microbenchmarks such as terasort and wordcount.

AMS-ERA adapts to changing conditions, improving load balancing and data locality in a way that traditional Hadoop resource allocation strategies, which tend to rely heavily on physical proximity, often fail to achieve. By dynamically selecting the best-suited nodes for each job, AMS-ERA reduces execution time and avoids resource contention. This innovative approach directly addresses the challenges of frugal clusters, where energy efficiency and resource constraints are paramount.

The rest of the paper is organized as follows. Section 2 presents relevant work and background. Section 3 details the proposed strategies and algorithms. Section 4 presents the extensive performance evaluation of the SBC cluster followed by Section 5, concluding this work.

2. Related Works

2.1. SBC in Cloud, Edge Clusters

One major challenge is the significant variance in computational capabilities among frugal nodes, which can lead to uneven workload distribution and resource contention. Frugal nodes typically have limited CPU processing power, memory, and storage, which can constrain the types and sizes of tasks they can effectively execute [17–19]. Moreover, these nodes may be deployed in edge or remote locations with unreliable network connectivity, posing challenges for communication and data transfer between nodes [20]. Additionally, frugal nodes are often deployed in resource-constrained environments where power consumption and energy efficiency are critical considerations. Balancing computational demands while minimizing energy consumption becomes crucial in such scenarios.

Shwe et al. [21] analyzed the efficacy of SBC-based clusters in three application scenarios. This work compares big data processing platforms across three computing paradigms—batch, stream, and function processing—in resource-constrained environments such as edge and fog computing, versus traditional cloud deployments. Using Apache Spark for batch processing, Apache Flink for stream processing, and Apache OpenWhisk for function processing, results suggest that resource-constrained environments can effectively handle big data workloads. The researchers provide recommendations for practical deployments in edge and fog computing and explore future research into training complex deep learning models in these environments.

In [3], Neto et al. outline the development, testing, and monitoring of a low-cost big data cluster using Raspberry Pi 4B devices [22] running Apache Hadoop. The results demonstrate that Raspberry Pi combined with Apache Hadoop can be a robust, cost-effective solution for a low-cost big data cluster. A Raspberry Pi 3B+ is used as a monitoring server to collect real-time data, thereby enabling improved monitoring and visualization of cluster performance. The authors of [23] examine the use of Apache Hadoop on a cluster of Raspberry Pi 4B SBCs to assess their potential as low-cost, energy-efficient platforms for big data processing. Through a series of benchmarks and different storage configurations, the research demonstrates that Raspberry Pi clusters can effectively handle large workloads, offering a viable alternative to traditional servers.

In [24], Lambropoulos et al. explore the use of SBCs like Raspberry Pi 4B for edge computing, demonstrating a successful transition from traditional x86 infrastructure to SBC-based clusters. Despite higher CPU usage and storage latency, the SBC-based setup showed significant power savings, consuming nine times less energy. The study discusses challenges with hardware compatibility and storage performance, suggesting future work on dedicated storage solutions and improved hardware customization to overcome limitations in edge environments. In [20], Sebbio et al. analyze the suitability of using a Raspberry Pi 4 device for federated learning as an edge device. They conduct a thorough power consumption analysis using the FedAvg algorithm for various datasets. Mills et al. [25] propose modifications to the FedAvg algorithm to address communication issues on an edge-computing-like testbed. The testbed is constructed using Raspberry Pi 2 and 3 clients composed of 10 devices.

In [26], Krpic et al. explore how SBC clusters handle compute-bound applications, using the High-Performance Linpack (HPL) benchmark to compare two four-node clusters of different SBC generations: Odroid U3 and Odroid MC1. Results indicate that SBC clusters can indeed serve as small-scale high-performance computing (HPC) systems, capable of managing moderate compute workloads at the edge. The authors of [27] propose a new Hadoop YARN architecture with two scheduling policies, namely master-driven and slave-driven. These are specifically designed for SBC Hadoop Clusters for big data processing. The authors design a small Hadoop cluster composed of Raspberry Pi 4 devices to validate the proposed policies. The authors of [28] construct a Raspberry Pi-based Hadoop cluster for image processing with various datasets of different sizes. They compare the computation time of the SBC cluster against a PC and note that the SBC cluster takes less time to complete the tasks for smaller datasets.

The above-mentioned works highlight the proposition of deploying Hadoop clusters in edge environments with SBCs like Raspberry Pi as a viable option [19–21,23–27], driven by cost-effectiveness, energy efficiency, sustainability, and flexibility. Although it may require addressing certain challenges, the benefits in terms of reduced latency, scalability, and sustainability make it a compelling choice for many edge and remote scenarios. While deploying Hadoop on SBC clusters offers advantages, it also introduces challenges such as hardware compatibility, performance issues, and software limitations. Addressing these issues may require customized hardware solutions, improved storage controllers, and adaptations to existing algorithms.

2.2. Hadoop YARN Scheduling Challenges in Resource-Constrained Clusters

Hadoop is an open-source framework designed for the distributed processing of large datasets across clusters of computers. At its core, the Hadoop Distributed File System (HDFS) serves as the distributed storage system, facilitating the reliable and scalable storage of data across the cluster. YARN (Yet Another Resource Negotiator) is a key component of the Hadoop ecosystem designed to manage cluster resources and allocate them for processing tasks. Its architecture consists of three main components: the Resource Manager (RM), Node Manager (NM), and Application Master (AM). The RM acts as the central authority for resource management, overseeing job scheduling and monitoring the overall cluster status. NM is responsible for managing resources on individual nodes in the cluster,

including monitoring resource usage and executing tasks. The AM is specific to each application and negotiates resources with the RM. YARN's flexibility and scalability enable it to efficiently manage resources across large-scale distributed computing environments. The RM consists of a Scheduler, which is responsible for allocating resources to different applications running on the cluster. The Application Manager oversees the lifecycle of applications submitted to the cluster. It coordinates with the NM to allocate resources, monitor application progress, and handle application-specific requests. Lastly, the ResourceTracker communicates with the NM to gather resource status and availability information from individual nodes in the cluster. This information is relayed to the Scheduler, facilitating efficient resource allocation.

Resource-aware and data locality-aware scheduling in Hadoop is a critical factor in improving performance, particularly in the context of big data processing. Efficient scheduling algorithms are crucial to ensure optimized cluster resource utilization, throughput, and fairness. In default Hadoop scheduling, several challenges concerning work distribution and balancing were reported by researchers. Inefficient data placement results in increased data transfer overhead and longer job execution times, impacting overall system efficiency. The scheduling mechanisms in Hadoop face several challenges, particularly in heterogeneous environments [6,29–32]. One issue is the lack of awareness of individual node capacities, including CPU processing power, memory availability, and disk storage. Consequently, tasks may be assigned to nodes that are ill equipped to handle them efficiently, leading to suboptimal performance and resource utilization. Another challenge arises from the dynamic and unpredictable nature of workloads in distributed environments. Traditional scheduling policies may struggle to adapt to changing workload patterns, resulting in inefficient resource allocation and potential bottlenecks [14]. Additionally, ensuring data locality, where computation is performed near the data it operates on, can be challenging in large-scale clusters with diverse hardware configurations. Inefficient data placement can lead to increased data transfer overhead and longer job execution times [10,33]. Addressing these problems requires the development of adaptive scheduling algorithms that can intelligently allocate resources based on workload characteristics and cluster dynamics while optimizing data locality and resource utilization.

Ullah et al. [32] introduced the Least Slack Time-Based Pre-emptive Deadline Constraint Scheduler (LSTPD) to enhance response and completion times for heterogeneous MapReduce jobs. They propose an efficient pre-emptive deadline constraint scheduler based on the least slack time and data locality. It first analyzes the task scheduling logs of the Hadoop platform; next it considers the remaining execution time of the job being executed in deciding pre-emption for scheduling. Javanmardi et al. [31] presented a unit-based, cost-efficient scheduler for heterogeneous Hadoop systems, focusing on running jobs in parallel on diverse clusters. The proposed algorithm distributes data based on the performance of the nodes and then schedules the jobs according to their cost of execution and decreases the cost of executing the jobs. The presented algorithm offers better performance in terms of execution time, cost, and locality compared to YARN-native FIFO and Fair schedulers.

Yao et al. [30] proposed new scheduling algorithms for Hadoop YARN clusters to improve performance and resource utilization, leveraging fine-grained resource management schemes to reduce the total execution time of MapReduce jobs. This is achieved through leveraging insights derived from requested resources, resource capacities, and task dependencies. In [29], Fu et al. propose a dynamic feedback load balancing scheduling method for fair task scheduling in Hadoop. An improved task scheduling strategy based on a genetic algorithm is proposed to allocate and execute application tasks to reduce task completion time. They also propose a delay capacity scheduling algorithm to ensure that most tasks can achieve localization and speed up job completion time. The researchers in [13] developed a novel job scheduler, CLQLMRS, using reinforcement learning to improve data and cache locality in MapReduce job scheduling, highlighting the importance of reducing job execution time for enhancing Hadoop performance. The limitations of

the study include the need to train the scheduling policy, which may be challenging in environments with rapid changes, potentially hindering timely retraining.

In [14], the authors propose a DQ-DCWS algorithm to balance data locality and delays in Hadoop while considering five Quality of Service factors. The DQ-DCWS is based on dynamic programming in calculating the length of the edge in the DAG and scheduling tasks along the optimal path. It aims to optimize workflow scheduling in data-intensive scientific applications on heterogeneous cloud resources. The authors evaluated their work using Montage workflow and deployed a Hadoop cluster over Amazon Elastic Compute Cloud (EC2). A Resource- and Network-aware Data Placement Algorithm (RENDA) in Hadoop is presented in [12]. The RENDA reduces the time of the data distribution and data processing stages by estimating the heterogeneous performance of the nodes on a real-time basis. It carefully allocates data blocks in several installments to participating nodes in the cluster. Experimental results show that RENDA outperforms recent alternatives in reducing data transfer overhead, average job completion time, and providing average speedup. RENDA's performance is largely dependent on the estimation of the remaining time of the nodes and subsequent data block distribution.

Postoaca et al. [15] presented a deadline-aware FOG-Scheduler for cloud edge applications. The job queue is ordered for context based on deadlines. The nodes in the cluster are ordered using a similarity index. The highest-ordered jobs are sorted and assigned to the appropriate clusters. The proposed algorithm is tested in Apache Spark. In [16], Vengadeswaran et al. propose an IDaPS based on intra-dependency (IDaPS) among data blocks to enhance performance and reduce data transfer overheads. The proposed IDaPS uses the Markov clustering algorithm to characterize MapReduce task execution based on intra-dependency and task execution frequency. Next, the scheduling algorithm uses the task execution frequency to determine a utility index. To achieve maximum parallelism, the jobs with the maximum utility index are assigned for execution.

In [34], Zhou et al. presented an adaptive energy-aware framework called AFED-EF for VM deployment in Cloud Data Centers. This framework aimed to address energy efficiency and SLA violations for IoT applications by considering variable loads. The proposed algorithm classifies the servers in the data center into various clusters using a K-means algorithm. Using this classification, the proposed algorithm determines the suitable server for load balancing. The study utilized real workload data from the CoMon project to evaluate the performance of the proposed algorithm. The results show that the proposed algorithm effectively balances energy consumption and SLA violations in data centers.

The authors of [35] address the issue of high energy consumption in Cloud Data Centers while minimizing Service-Level Agreement (SLA) violations. To achieve this, the study proposed two adaptive energy-aware algorithms aimed at maximizing energy efficiency and reducing SLA violation rates. The proposed algorithms considered application types, CPU, and memory resources during VM deployment. In [36], Banerjee et al. present a Dynamic Heuristic Johnson Sequencing technique (DHJS) for job scheduling in Hadoop. They apply the proposed technique to Hadoop default scheduling algorithms to determine the best order of jobs on each server, thereby minimizing the makespan. The experimental results presented show performance improvement; however, the results are based on testing using only three servers with limited scope.

Table 1 summarizes the contributions of recent studies, providing an overview of various heterogeneous cluster scheduling techniques. Each work is categorized based on its research focus, whether it uses resource-aware scheduling techniques that consider CPU, memory, disk, or network resources in decision-making, and the type of testbed deployment, whether it involves servers or Single-Board Computer (SBC) clusters. The table also indicates the evaluation criteria, noting whether Hadoop microbenchmarks or custom datasets were used for performance assessment.

Table 1. Overview of heterogeneous cluster scheduling techniques.

Category	Representative Works	Resource Awareness	Testbed/Evaluation Criteria
Task placement	[9]: Task scheduling based on network heterogeneity	net	X
	[12]: RENDA—Estimation of node performance for task placement	CPU, mem	Servers/Benchmarks
	[36]: Dynamic Heuristic Johnson Sequencing technique for task placement	X	Simulation
	[14]: DQ-DCWS—Optimization of workflow using dynamic programming	disk	Simulation
	[30]: HASTE: Resource management for improved task placement	CPU, mem	Servers/Benchmarks
Data locality	[10]: Varying Block size for improved data locality	disk	Servers/Benchmarks
	[27]: Resource-aware task placement in heterogeneous SBC clusters	CPU, mem, disk	SBC cluster/Benchmarks
	[13]: CLQLMRS—Reinforcement learning improves data locality	disk, mem	Servers/Benchmarks
Load balancing	[11]: Historical data-based task placement in heterogeneous clusters	CPU, mem	Servers/Benchmarks
	[15]: Deadline-aware task scheduling based on available resources	X	Servers/Custom dataset
	[29]: Dynamic feedback fair scheduling with load balancing	X	Servers/Benchmarks
Improved parallelism	[16]: Markov clustering-based job scoring for improved task allocation	disk	Servers/Benchmarks
	[31]: Optimizing DAG workflows for the cost of task execution	disk	Simulation
	[32]: LSTPD—Deadline-constrained response times for MapReduce jobs	X	Servers/Benchmarks
Improved task selection	[37]: Task selection using K-means clustering technique	X	X
	[38]: H-Fair; improved Fair scheduler for heavy workloads in Hadoop	X	Simulation
	[39]: Improved MapReduce workflow using K-means clustering	X	Servers/Custom dataset
Energy efficiency	[33]: Efficient online placement in cloud containers	X	Simulation
	[34]: AFED-EF—Classification of resources based on energy efficiency	CPU, mem, disk, net	Server/Real workload
	[35]: Energy-efficient scheduling based on resource constraints.	CPU	Servers/Custom dataset

Overall, scheduling tasks in heterogeneous clusters composed of frugal nodes require specialized optimization techniques and adaptive scheduling algorithms tailored to the unique characteristics and constraints of these devices. In this work, we propose AMS-ERA for resource-aware scheduling in Frugal Heterogeneous Hadoop Clusters. Our criterion considers CPU, memory, and disk requirements for jobs and aligns the requirements with available resources in the cluster for optimal resource allocation. We construct an 11-node SBC cluster to test and validate the proposed approach using a Hadoop benchmark for CPU-intensive and IO-intensive workloads. The next section presents details for the proposed AMS-ERA.

3. Adaptive Multi-Criteria Selection for Efficient Resource Allocation

3.1. Motivation

The native Hadoop framework lacks a built-in mechanism to distinguish the specific capacities of individual nodes, including CPU processing power and available physical memory and storage availability. Such characteristics are crucial in edge clusters composed of resource-frugal devices, as they significantly influence the performance of concurrently executing MapReduce tasks.

Consider a scenario in which a cluster consisting of N nodes must process M MapReduce tasks across D data blocks. According to the default configuration of Hadoop's

InputSplit, the number of MapReduce tasks corresponds to the number of data blocks, meaning each task operates on one data block per node. However, this default approach overlooks the available utilization of resources within the cluster leading to the suboptimal allocation of resources. Typically, a node can have multiple CPU cores available, and an optimal resource allocation strategy can leverage the available resources to allow the simultaneous execution of multiple MapReduce jobs to improve the parallelism in the cluster, thereby improving the overall efficiency of the cluster.

Single-Board Computers (SBCs), exemplified by the Raspberry Pi computers, typically feature quad-core processors, with more advanced models boasting hexa- or octa-core processors. Leveraging these resources effectively for optimal resource allocation is crucial. Additionally, SBCs have limited onboard memory and disk capacity. In many instances, the default Hadoop input split may not allocate data blocks optimally on these SBC-based nodes, resulting in various out-of-memory errors [3,23]. Consequently, the MapReduce jobs fail and need to restart, which can be expensive. In Table 2, we present a matrix listing the various features of popular SBCs.

Table 2. A comparison of popular Single Board Computers.

SBC Device	CPU	Memory	Storage with Read MB/s	Price (USD) Incl. Storage
Raspberry Pi3 [22]	1.4 GHz 64-bit quad-core ARM Cortex-A53	1 GB LPDDR3-SDRAM	32 GB SD Card 120 MB/s	38
Odroid Xu4 [40]	Exynos5 Octa ARM Cortex-A15 Quad 2 GHz and Cortex-A7 Quad 1.3 GHz	2 GB DDR3	32 GB SD Card 120 MB/s	56
RockPro64 [41]	1.8 GHz Hexa Rockchip RK3399 ARM Cortex A72 and 1.4 GHz Quad Cortex-A53	4 GB LPDDR4-SDRAM	64 GB SD Card 140 MB/s	84
Raspberry Pi4 × 4	1.8 GHz Quad core ARM Cortex-A72	4 GB LPDDR4-SDRAM	64 GB SD Card 140 MB/s	59
Raspberry Pi4 × 8	1.8 GHz Quad core ARM Cortex-A72	8 GB LPDDR4-SDRAM	128 GB SD Card 190 MB/s	84
Raspberry Pi5	2.4 GHz Quad-core 64-bit ARM Cortex A76	8 GB LPDDR4X-SDRAM	128 GB SD Card 190 MB/s	98

Moreover, the positioning of data blocks on nodes where MapReduce tasks are executed is crucial for efficient processing, aiming to minimize latency in data transfers between different nodes within the cluster. Given the limited available resources on the frugal SBC-based Hadoop clusters, it is essential to develop optimal resource allocation strategies tailored to frugal Hadoop clusters, considering the unique resource constraints of SBCs. Table 3 lists the main symbol notations and their meanings used in this paper.

Table 3. Symbols used and their meanings.

Symbol	Description
$J = \{j_1, \ldots, j_k\}$	Set J of Jobs consisting of k number of jobs
$C = \{n_1, \ldots, n_x\}$	Cluster C consisting of x number of nodes
id_i	Unique identifier for a job j_i
cpu_{ri}	CPU requirement for job j_i
$disk_{ri}$	Disk requirement for job j_i
mem_{ri}	Memory requirement for job j_i
$U(cpu_i)$	CPU utilization of ith node
$U(mem_i)$	Memory utilization of ith node
$U(disk_i)$	Disk utilization of ith node
$resource_{list}$	A data structure detailing the available resources in the cluster

Table 3. *Cont.*

Symbol	Description
$k_{optimal}$	Optimal value for centroids in K-means algorithm
μ_j	Centroid in K-means clustering algorithm
Cji	Pairwise decision criteria matrix
CI	Consistency Index
λ_{max}	Maximal eigenvalue
$weight_i$	Weight of CPU, mem, and disk matrices
m_{ij}	Normalized scores
$Score_i$	Score for job j_i
l_i	Workload I

3.2. Problem Definition

We define a few terms to quantify the proposed research. We assume that a set of k number of jobs $J = \{j_1, \ldots, j_k\}$ is submitted to a heterogeneous Hadoop cluster consisting of x number of nodes $C = \{n_1, \ldots, n_x\}$.

As each job may have unique CPU, memory, disk, and I/O requirements, we model a vector consisting of these parameters for a job j_i,

$$j_i = \{id_i, cpu_{ri}, disk_{ri}, mem_{ri}\} \tag{1}$$

where cpu_{ri} is the CPU, $disk_{ri}$ is the disk, and mem_{ri} is the memory requirement for the job j_i with a unique id_i.

To define the utilization $U = \{U(CPU),\ U(mem), U(disk)\}$ of resources available in a node n_i in Cluster C at time t, we give

$$U(cpu_i) = \frac{\{100 - (\%\ of\ idle\ time)\}}{100} \tag{2}$$

where $U(cpu_i)$ is the CPU utilization of the ith node. The memory utilization $U(mem_i)$ of a node n_i is given as

$$U(mem_i) = \frac{\sum mem_k}{mem(total)} \tag{3}$$

where $\sum mem_k$ is the sum of memory usage of all jobs running in node n_i where $mem(total)$ is the total memory available on the node. The disk utilization $U(disk_i)$ of the ith node is given as

$$U(disk_i) = \frac{disk(used)}{disk(total)} \tag{4}$$

where $disk(used)$ is the used capacity and $disk(total)$ is the total disk capacity of the ith node. The values of utilization are within range [0, 1].

In the Hadoop YARN cluster architecture, the NMs within Cluster C regularly transmit status updates as heartbeat messages to the RM. These messages convey crucial information regarding resource availability, including CPU utilization, memory usage, and disk I/O activity for a data node managed by the corresponding NM.

The Hadoop cluster's execution traces can be obtained using the *Starfish* Hadoop log analysis tool [42], serving as crucial input for refining data placement decisions. Throughout the execution of each job within the cluster, essential details such as Job ID and job timestamp are captured and stored as job status files. These execution traces are typically located in the configuration directory of the name node. The location of this file is available in the Hadoop name node job history folder.

The proposed AMS-ERA leverages machine learning techniques such as K-means clustering to cluster similar data by grouping nodes into clusters based on their similarity or proximity to each other. We use these techniques to classify nodes in the cluster based on the similarity of utilized resources (*CPU*, *mem*, *disk*), initializing a $resource_{list}$ that is

subsequently used by the dAHP. A schematic diagram of various steps in the AMS-ERA process can be seen in Figure 1.

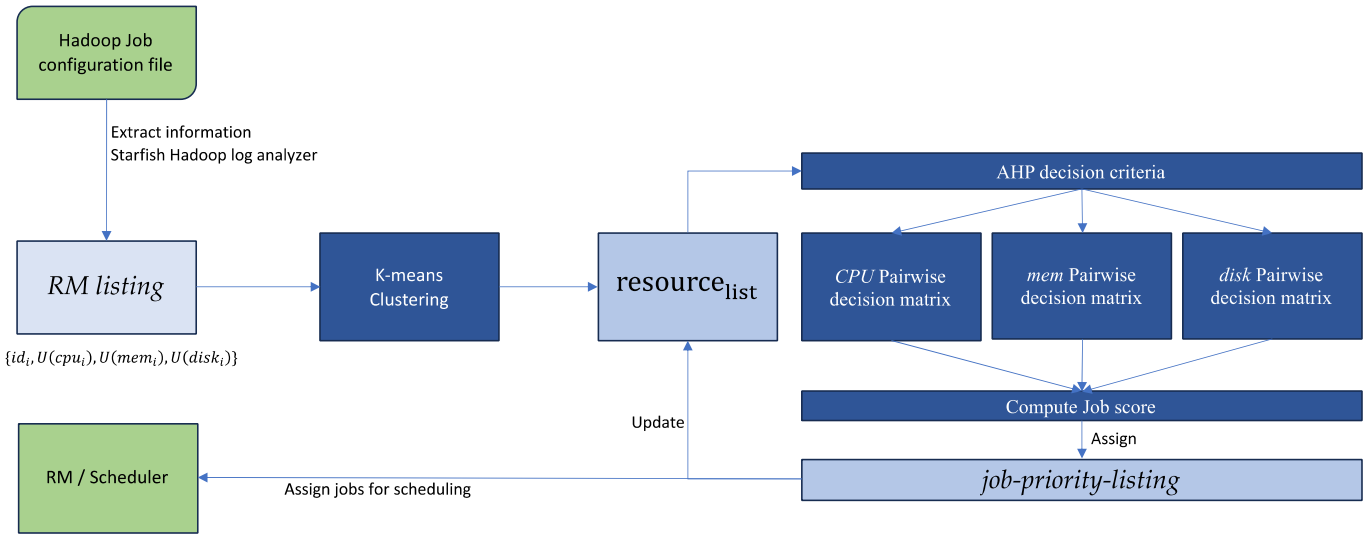


Figure 1. Workflow of the proposed AMS-ERA process for optimal job scheduling.

3.3. K-Means with Elbow Clustering

K-means clustering is a popular unsupervised machine learning algorithm used in various domains. It is used to analyze, optimize, and manage resources, among other applications [37]. K-means partitions a dataset into k clusters based on their features. It selects k centroids, and each datapoint is assigned to the closest centroid. The elbow method [43] is used to determine the optimal number of clusters $k_{optimal}$. It involves plotting the within-cluster sum of squares (WCSS) for different values of k and identifying the "elbow" point where increasing the number of clusters does not significantly reduce the WCSS, indicating the optimal k value.

Algorithm 1 presents the proposed K-means with an elbow optimization algorithm. It starts by obtaining the RM listing for n nodes. Next, we use Min–Max normalization [44] to rescale numerical data from the Hadoop RM to a fixed range. This normalization method preserves the relative relationships between datapoints while ensuring that all features have the same scale. We define the dataset $D = \{x_1, x_2 \ldots x_n\}$, where each x_i is a datapoint. We verify the status of all the nodes to remove any nodes that have a failed state. In the node identification phase, based on the acquired parameters (*U(cpu)*, *U(mem)*, *U(disk)*), the proposed approach organizes nodes into clusters characterized by similar performance attributes.

Next, we determine the $k_{optimal}$ where k is initially given in a range of 1 to k_{max}. First, we select k initial centroids given by $\mu_1, \mu_2 \ldots \mu_k$ via random selection. Next, we assign each datapoint x_i to the nearest centroid μ_i. We then define C_j as the set of datapoints assigned to the jth cluster:

$$C_j = \{x_i \mid \|x_i - \mu_j\|^2 \leq \|x_i - \mu_p\|^2 \; for \; all \; p = 1, 2, \ldots, k\} \tag{5}$$

where $\|x_i - \mu_j\|^2$ represents the squared Euclidean distance. Next, we recalculate each centroid μ_j as the mean of the datapoints in its cluster, given as follows:

$$\mu_j = \frac{1}{|C_j|} \sum_{x \in C_j} x \tag{6}$$

We repeat steps in Equations (5) and (6) until the centroids no longer change significantly or a predetermined number of iterations is reached. The WCSS for the clustering with k clusters is given as follows:

$$WCSS = \sum_{j=1}^{k} \sum_{x \in C_j} \|x_i - \mu_j\|^2 \tag{7}$$

Next, we plot the WCSS against k and identify the "elbow" point, where the reduction in WCSS starts to plateau. This elbow point suggests the optimal number of clusters $k_{optimal}$ for the dataset D.

Once the $k_{optimal}$ is determined, we use the K-means clustering algorithm to cluster the nodes based on resource utilization. Each datapoint is assigned to the nearest centroid μ_p. Next, we calculate the Euclidean distance of datapoint x_i to each centroid μ_p. The datapoint xi is assigned to the nearest centroid based on the closest distance to the selected centroid. After all datapoints have been assigned, we recalculate each centroid as the mean (average) of all the datapoints assigned to that cluster. The process for the recalculation of centroids is repeated $k_{optimal}$ times. Once the node similarity clusters are established, our strategy organizes the groups based on the three selection attributes *CPU*, *mem,* and *disk*, with higher-performing nodes belonging to higher-ranked clusters. The resulting data are written to $resource_{list}$ for further processing. The runtime for Algorithm 1 can be given as $O(k \times x)$ where x is the number of servers/nodes in the cluster C; k is the number of clusters.

Algorithm 1: K-means clustering with elbow

Start: Obtain RM listing for n nodes.
    apply Min–Max normalization to rescale the dataset
    initialize $\text{resource}_{\text{list}} \leftarrow \{id_i, U(cpu_i), U(mem_i), U(disk_i)\}$
    Let $D = \{x_1, x_2 \ldots x_n\}$ be the dataset, where x_i is a datapoint
    determine $k_{optimal}$ for K-means using Equations (5)–(7)
    foreach k in $\{1, 2, \ldots k_{optimal}\}$
        calculate the distance of each datapoint x_i to each centroid μ_p
        assign each datapoint x_i to the closest centroid
        recalculate each centroid as in Equation (6)
    end for
    return $\text{resource}_{\text{list}}$
end

3.4. Dynamic AHP-Based Job Scoring

In this section, we detail the dynamic AHP-based scoring mechanism for optimal resource allocation to jobs. We develop an algorithm based on the AHP [45], where the goal is to find the optimal placement of a job using a score vector to determine the best possible node. The process involves refining the AHP model's accuracy by integrating historical information obtained through Hadoop APIs to assign weights to jobs based on their resource requirements including CPU, memory, and disk requirements.

We define criteria considering the CPU, memory, and disk requirements of a job. We also define alternate criteria for selecting the best possible node in the frugal cluster. The criteria are pairwise compared based on the importance of the criteria. The alternatives are compared against each of the criteria. Figure 2 shows the selection framework of the dAHP for an example of n heterogeneous nodes, where $n = 6$.

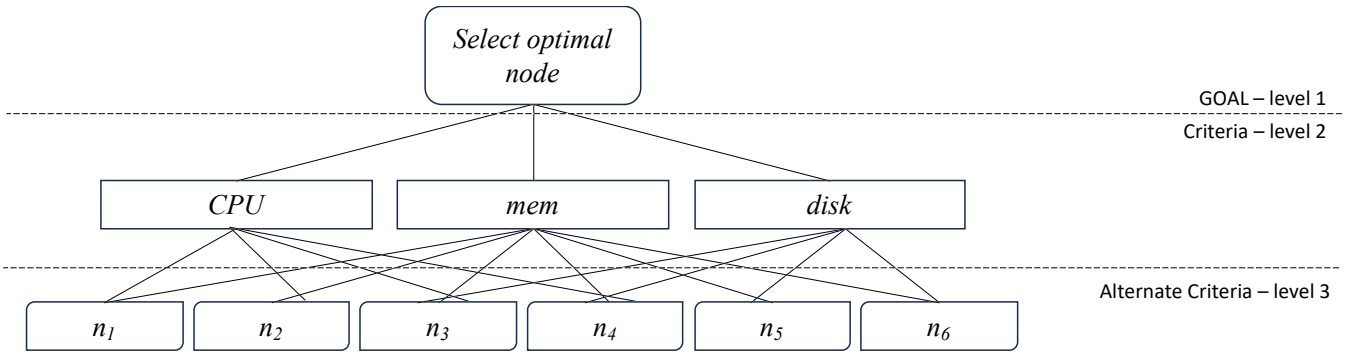


Figure 2. dAHP 3-level criteria for node selection.

To select the optimal node for job allocation, we look at the job requirements, assuming that a job requires a large amount of processing power and memory to complete; however, the storage requirement is not equally important. Based on this requirement, we develop the pairwise comparison matrix *Cji* for this job. The criteria are prioritized based on their importance. We assume that the CPU and memory requirements are equally important for a job. They are moderately more important than the disk requirement. Based on these criteria, the *Cji* is given in Table 4.

Table 4. Pairwise decision criteria matrix C*ji*.

Cji	CPU	mem	disk
CPU	1	1	2
mem	1	1	2
disk	½	½	1

Next, the alternate criteria are considered based on the node utilization requirements. For instance, if a job requires a faster node, it should be assigned n_2. If it requires more memory, it can be assigned n_6. Similarly, if more disk space is required, it can be assigned n_6 based on the node capabilities. The weights are determined through the magnitude of the difference in node properties. For instance, if n_2 and n_5 have a faster/larger number of cores compared to n_1, they are assigned weight 4. This allows us to have three matrices presenting a pairwise comparison of CPU, MEM, and DISK requirements. The CPU, MEM, and DISK matrices are given in Tables 5–7, respectively.

Table 5. Pairwise *CPU* alternate criteria decision matrix CPU.

CPU	n_1	n_2	n_3	n_4	n_5	n_6
n_1	1	0.25	0.33	0.33	0.25	0.25
n_2	4	1	3	2	2	1
n_3	3	0.33	1	1	1	0.5
n_4	3	0.5	1	1	1	0.5
n_5	4	0.5	1	1	1	0.5
n_6	4	1	2	2	2	1

Table 6. Pairwise *mem* alternate criteria decision matrix MEM.

MEM	n_1	n_2	n_3	n_4	n_5	n_6
n_1	1	0.5	0.33	0.33	0.25	0.25
n_2	2	1	0.5	0.5	0.33	0.33
n_3	3	2	1	1	0.5	0.5

Table 6. *Cont.*

MEM	n_1	n_2	n_3	n_4	n_5	n_6
n_4	3	2	1	1	0.5	0.5
n_5	4	3	2	2	1	0.5
n_6	4	3	2	2	2	1

Table 7. Pairwise *disk* alternate criteria decision matrix DISK.

DISK	n_1	n_2	n_3	n_4	n_5	n_6
n_1	1	1	0.5	0.5	0.33	0.33
n_2	1	1	0.5	0.5	0.33	0.33
n_3	2	2	1	1	0.5	0.5
n_4	2	2	1	1	0.5	0.5
n_5	3	3	2	2	1	1
n_6	3	3	2	2	1	1

To ensure consistency and accuracy, the pairwise comparison matrices are normalized to determine the Consistency Index (CI). A matrix is considered to be consistent if the transitivity rule is valid for all pairwise comparisons [39]. The CI is determined via

$$CI = \frac{\lambda_{max} - n}{n - 1} \tag{8}$$

where λ_{max} is the maximal eigenvalue obtained via the summation of products between each element of the eigenvector and the sum of columns of the matrix and n is the number of nodes. In this case, since the size of the matrix is 6×6 where $n = 6$, the CI value for each of the CPU, MEM, and DISK is 0.1477, 0.0178, and 0.0022, respectively. Using the Random Index (RI) = 6, the Consistency Ratio (CR) for each of these matrices is 0.0119, 0.0143, and 0.0017, respectively. For reliable results, the CR values must be less than 0.1, ensuring that the matrices are consistent. The score vector $Score_i$ is determined for the job j_i using the **M** matrix, which consolidates the CPU, MEM, and DISK matrices, as given in Equation (6).

$$Score_i = max \sum_{j=1}^{n} weight_i \,.\, m_{ij} \tag{9}$$

where $weight_i$ is the weight of *CPU*, *mem*, and *disk* obtained from corresponding matrices for the defined criteria and m_{ij} is the normalized score for each value in the matrix. The score vector $Score_i$ computed in Equation (6), is given in Table 8. In this particular case, n_6 has the highest score = 0.280, indicating that it is the most suitable for the given requirements of job j_i.

Table 8. The *score* vector determined from the M matrix for every alternative.

M	Weight	n_1	n_2	n_3	n_4	n_5	n_6
CPU	0.4	0.051	0.278	0.130	0.138	0.146	0.258
mem	0.4	0.056	0.088	0.152	0.152	0.244	0.307
disk	0.2	0.082	0.082	0.149	0.149	0.270	0.270
score	-	**0.059**	**0.163**	**0.143**	**0.146**	**0.210**	**0.280**

Similar to this example, each job's score is determined using these criteria. After computing scores for all jobs, the system selects the job with the highest score, indicating the greatest resource demand. The resulting job priority list sorted in descending order (ordered by score) is forwarded to the RM for resource allocation.

3.5. Efficient Resource Allocation

The resource allocation takes place in the RM once the job priority listing is available. By integrating the job listing information obtained from the previous phase, the RM ensures the optimal match between job demand and available resources. The jobs with the highest resource requirements are arranged in descending order. These high-resource-demanding jobs are prioritized to utilize the most powerful nodes with the maximum available resources. This load-balancing strategy ensures that less-resource-intensive jobs do not hinder the utilization of the high-resource nodes.

Algorithm 2 presents the AMS-ERA resource allocation process. The RM maintains a $resource_{list}$ of current resource utilization in the cluster for each node I $\{id_i, U(cpu_i), U(mem_i), U(disk_i)\}$ as determined in Equations (2)–(4). To assign a job to a node, considering the jobs score vector {*cpu*, *mem*, *disk*} obtained in Table 8, the most under-utilized node is sought. Once a job is assigned to a node, the utilization values in the $resource_{list}$ for the corresponding node are updated. This ensures a well-balanced strategy that maximizes the utilization of resources across the cluster while considering the heterogeneous Hadoop cluster node capabilities. This enables our system to prevent resource-intensive jobs from being allocated to lower-performing nodes in the cluster. Once the mapping is complete, the jobs are sent for execution in newly allocated containers by YARN.

To give the runtime for the algorithm, we look at the three computation-intensive operations. Step 5 in the algorithm requires the computation of pairwise decision matrices for each job as detailed in Tables 5–7. Assuming that there are m jobs in a cluster of size n nodes, the cost of forming pairwise comparison matrices for the three selection criteria *CPU*, *mem*, and *disk is* $\frac{n(n-1)}{2}$, giving a complexity of $O(n^2)$. In step 6, we determine the values of Equations (8) and (9) for the normalization and consistency check; these require a runtime of $O(n)$. Steps 10 and 11 compute the M matrix and the score vector. The total number of pairwise comparisons required to compute the matrix M and the score vector would be given as $O(m \times n^2)$, where the size of the matrix is $n \times n$. Finally, the best values are written to the $\text{resource}_{\text{list}}$. Overall, the complexity of both algorithms can be given as $O(m \times n^2)$.

Algorithm 2: AMS-ERA resource allocation

1: **Start**: Obtain RM listing for n nodes
2: obtain $\text{resource}_{\text{list}}$
3: obtain RM listing for m *jobs*; initialize $job_{priority_{listing}}$
4: **foreach** $m \in jobs$
5: determine pairwise decision matrices CPU, MEM, $DISK$ for m
6: determine consistency $CI = \frac{\lambda_{max}-n}{n-1}$
7: **if** $CR = \frac{CI}{RI} < 0.1$ **then** continue
8: **else** re-compute
9: **end if**
10: determine **M** matrix
11: Compute $Score_m \rightarrow job_{priority_{listing}}$
12: **end for**
13: **foreach** $i \in job_{priority_{listing}}$
14: assign best $(j_i, cpu, mem, disk) \rightarrow \text{resource}_{\text{list}}$
15: update $\text{resource}_{\text{list}}$
16: **end for**

4. Experimental Evaluation

In this section, we present the experimental setup and conduct various experiments to compare and analyze the performance of the proposed AMS-ERA against Hadoop-Fair, FOG, and IDaPS Schedulers.

4.1. Experiment Setup

For experimentation, we construct an SBC-based heterogeneous Hadoop cluster with 11 SBC nodes configured in two racks with 5 SBCs in each rack using Gigabit Ethernet. Ten of these SBCs run the Hadoop worker nodes, whereas one serves as the master node. As the master node runs the RM, requiring a large amount of memory, a Raspberry Pi5 device is dedicated to running the master node. On each device, we install a compatible version of Linux Debian; with armbian 23.1 Jammy Gnome on RockPro, Debian Bullseye 11 on Odroid XU4, and RaspberryPi OS Lite 11 on all Raspberry Pi devices. Each device has Java ARM64 version 8 and Hadoop version 3.3.6. Each SBC is equipped with a bootable SD Card; to better observe the placement of jobs with different disk requirements, we varied the capacity of the SD Card for the different SBCs. A 4 GB swap space was reserved on each SD Card; this would be essential for virtual memory management in SBCs with low RAM availability.

To simulate a small cluster, we created two racks, each consisting of five SBC nodes. Each rack has a Gigabit Ethernet switch connecting all the SBCs with a router. The master node running on an Rpi5 connects to the router. The schematic diagram of the experimental setup is available in Figure 3. Table 9 shows the configuration of the worker nodes in the cluster. We used Hadoop YARN 3.3.6 to run our experiments. The HDFS block size was set to 128 MB, with block replication set to 2, and the *inputSplit* size was set to 128 MB. To avoid out-of-memory errors on Hadoop runs, we modified the mapred-site.xml and YARN-site.xml files. The details are provided in Table 10.

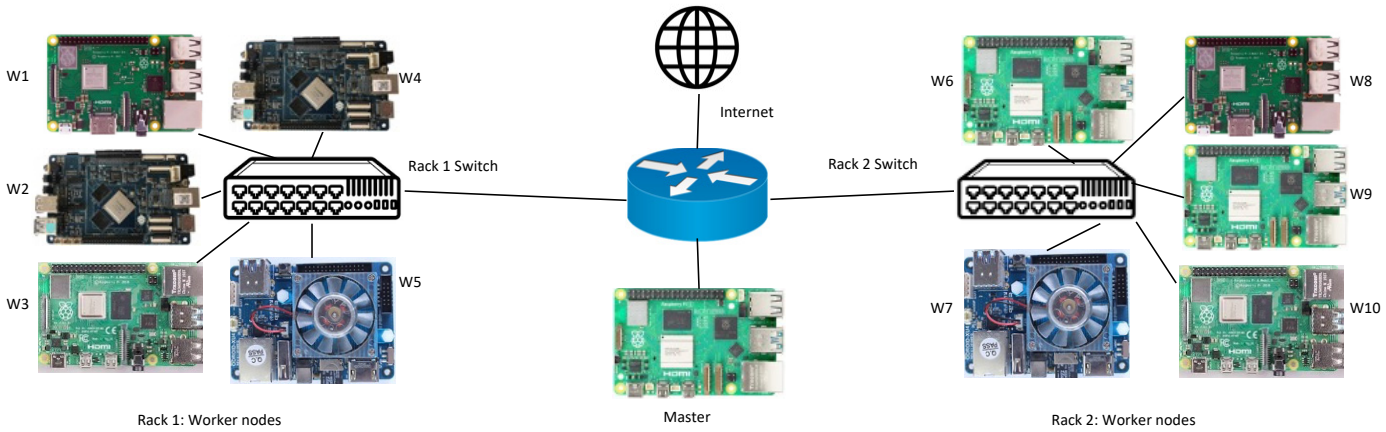


Figure 3. Cluster configuration with 10 worker nodes placed in two racks with a master node.

Table 9. Worker node configuration in the Hadoop cluster.

Worker Node	Rack	SBC Device	CPU Cores	Memory	Storage with Read MB/s	Operating System
W1	1	RaspberryPi3	4 (1.4 GHz)	1 GB	32 GB SD Card 120 MB/s	RaspberryPi OS Lite 11
W2	1	RockPro64	6 (2 × 1.8 GHz) (4 × 1.4 GHz)	4 GB	64 GB SD Card 140 MB/s	armbian 23.1 Jammy Gnome
W3	1	RaspberryPi4	4 (1.8 GHz)	4 GB	64 GB SD Card 140 MB/s	RaspberryPi OS Lite 11
W4	1	RockPro64	6 (2 × 1.8 GHz) (4 × 1.4 GHz)	4 GB	32 GB SD Card 120 MB/s	armbian 23.1 Jammy Gnome
W5	1	Odroid Xu4	8 (4 × 2.0 GHz) (4 × 1.3 GHz)	2 GB	64 GB SD Card 140 MB/s	Debian Bullseye 11
W6	2	RaspberryPi5	4 (2.4 GHz)	8 GB	128 GB SD Card 190 MB/s	RaspberryPi OS Lite 11
W7	2	Odroid Xu4	8 (4 × 2.0 GHz) (4 × 1.3 GHz)	2 GB	32 GB SD Card 120 MB/s	Debian Bullseye 11
W8	2	RaspberryPi3	4 (1.4 GHz)	1 GB	64 GB SD Card 140 MB/s	RaspberryPi OS Lite 11
W9	2	RaspberryPi5	4 (2.4 GHz)	8 GB	64 GB SD Card 140 MB/s	RaspberryPi OS Lite 11
W10	2	RaspberryPi4	4 (1.8 GHz)	4 GB	128 GB SD Card 190 MB/s	RaspberryPi OS Lite 11

Table 10. Hadoop YARN configuration properties.

Mapred-site.xml	Value
yarn.app.mapreduce.am.resource.mb	852
mapreduce.map.cpu.vcores	2
mapreduce.reduce.cpu.vcores	1
mapreduce.map.memory.mb	852
mapreduce.reduce.memory.mb	852
YARN-site.xml	**Value**
yarn.nodemanager.resource.memory-mb	1024
yarn.nodemanager.resource.cpu-vcores	1
yarn.scheduler.maximum-allocation-mb	1024
yarn.scheduler.maximum-allocation-vcores	8
yarn.nodemanager.vmem-pmem-ratio	2.1

4.2. Generating Job Workloads for Validation

Taking inspiration from previous benchmark studies [10,12,16,18,27,29,30,38], we select wordcount and terasort workloads for the evaluation of AMS-ERA.

- The Hadoop wordcount benchmark is a CPU-intensive task because it involves processing large volumes of text data to count the occurrences of each word. This process requires significant computational resources, particularly for tasks like tokenization, sorting, and aggregation, which are essential steps in the word-counting process. As a result, the benchmark primarily stresses the CPU's processing capabilities rather than other system resources such as memory or disk I/O. These 10 jobs are posted to the cluster simultaneously.
- The Hadoop terasort benchmark is an IO-intensive task because it involves sorting a large volume of data. This process requires substantial input/output (IO) operations as it reads and writes data to and from storage extensively during the sorting process. The benchmark stresses the system's IO subsystem, including disk read and write speeds, as well as network bandwidth if the data are distributed across multiple nodes in a cluster.

In order to observe the effectiveness of the proposed AMS-ERA scheduling for clustering the jobs based on the *CPU*, *mem*, and *disk* criteria, we generate five job workloads $\{l_1, l_2, l_3, l_4, l_5\}$, each with varying resource requirements, resulting in highly heterogeneous container sizes for the map and reducing tasks across different jobs. Each workload is given a different dataset whose sizes are 2, 4, 8, 15.1, and 19.5 GB, respectively. The datasets are generated from text files available at project Gutenberg. These dataset sizes represent a range of small to large workloads, allowing us to evaluate the scheduling algorithm's performance across different job scenarios. By including a range of dataset sizes, we could determine how the proposed AMS-ERA scheduling algorithm handles different resource requirements.

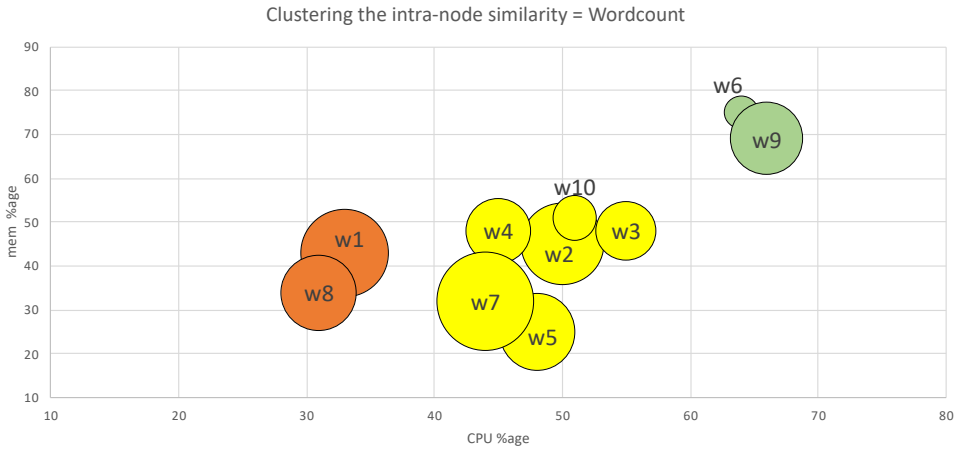
The default InputSplit size of 128 MB is used to distribute the datafiles across the HDFS. The replication factor of 2 is used. Based on the dataset size and the InputSplit size, we define the number of maps and reduces to be <map, reduce> given as <16, 1>, <32, 2>, <64, 4>, <128, 4>, and <160, 8>, respectively.

We execute wordcount and terasort on these workloads with these parameters and observe job placement, resource utilization, and the overall job execution time in the cluster. To ensure the reliability and robustness of our experimental study, we conducted multiple experimental runs for each benchmark and workload. Specifically, for each of the workloads (l_1 through l_5), we performed at least three experimental repetitions to gather consistent data. This repetition allowed us to account for any variability in cluster performance and ensure that our conclusions were statistically valid. Each experiment was run under the same conditions to maintain consistency, providing a strong basis for comparison across different configurations.

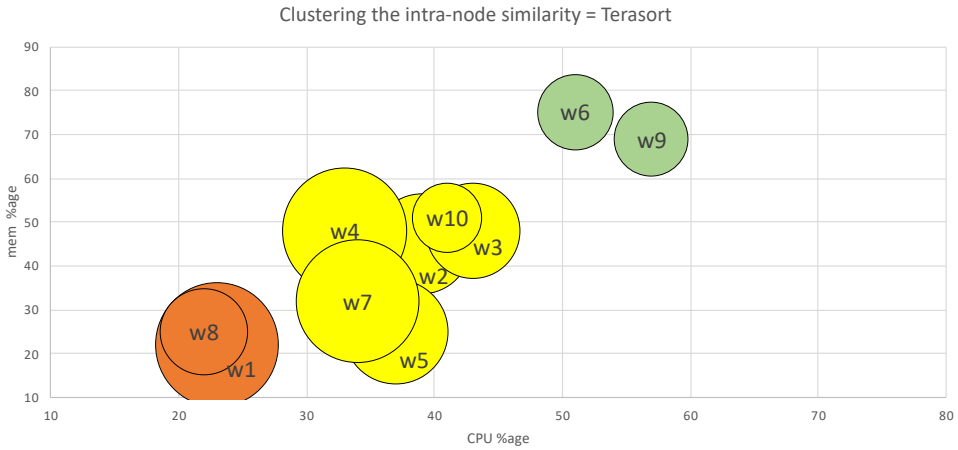
4.3. Node Clustering Based on Intra-Node Similarity Metrics

The AMS-ERA profiles the nodes available in the cluster based on available resources. We visually determined the elbow point for three experimental runs for workload l_3 using wordcount and terasort. Workload l_3, with a dataset size of 8 GB, served as a suitable test case for determining the optimal value for k. This dataset is large enough to offer significant insight into node resource clustering while not being so large as to skew results due to extreme data processing demands. Moreover, by establishing $k_{optimal}$ for this workload, the same methodology can be applied to smaller workloads (l_1 and l_2) or larger workloads (l_4 and l_5), ensuring that the clustering approach can be scaled effectively based on the size and complexity of the data being processed. Based on these experiments, we determine the value of $k_{optimal}$= 3.

Figure 4a reveals the result of AMS-ERA node grouping based on available resources during the execution of the wordcount benchmark using the workload l_3. A high-performance group of nodes is highlighted in green, consisting of nodes w6 and w9; a medium-performance group of nodes is represented in yellow, comprising w2, w3, w4, w5, w7, and w10. The nodes labeled low-performance are indicated in orange, including nodes w1 and w8.



(**a**)



(**b**)

Figure 4. The worker nodes profiling based on their CPU, mem, and disk resource utilization for workload l_3. Intra-node similarity reveals the performance of nodes clustered in high-, medium-, and low-performance nodes for wordcount jobs. The size of the bubble reveals the percentage of disk utilization. (**a**) Wordcount and (**b**) terasort.

Similarly, Figure 4b reveals the results for terasort jobs run for workload l_3; comparatively, terasort requires large disk IO. The proposed clustering algorithm successfully groups the nodes based on the utilization of CPU, mem, and disk resources for terasort jobs with different requirements. Worker nodes w1, w4, and w7 are installed with relatively smaller 32 GB and slower disk IO read/write speed SD Cards. As the terasort progresses, more of their onboard available storage is consumed. Consequently, nodes w6, w9, and w10 give identical disk IO, which is comparatively slower due to the user of faster hardware. The effect of larger disk IO can be seen in Figure 4b, where a larger bubble area reveals increased disk IO.

4.4. Workload Execution Time

Hadoop Fair Scheduler (Fair), a default scheduler in Hadoop, lacks data locality implementation. It allocates resources to jobs, ensuring each job receives an equal share of resources over time. The scheduler organizes jobs into pools and distributes resources equitably among these pools.

The FOG-scheduler presented in [15] considers ordering the scheduling queue based on deadlines. The nodes in the cluster are ordered using a similarity index. The highest-ordered jobs are sorted and assigned to the appropriate clusters.

The IDaPS presented in [16] uses the Markov clustering algorithm to characterize MapReduce task execution based on intra-dependency and task execution frequency. The scheduling algorithm orders the tasks based on execution frequency to achieve maximum parallelism. In this section, we compare these recent works against the proposed AMS-ERA for various workloads of wordcount and terasort.

Figure 5a shows the comparison of the execution time of the five wordcount workloads using Hadoop default fair scheduler, FOG, IDaPs, and the proposed AMS-ERA. For smaller workloads l_1 and l_2, a total of 16 and 32 map jobs are created; with this workload, the execution runtimes for the proposed AMS-ERA are 27.2%, 17.4%, and 7.6% and 24.5%, 14.1%, and 8.1% faster compared to Fair, FOG and IDaPs, respectively.

For workloads l_3 and l_4, a total of 64 and 128 map jobs were created; the execution times for AMS-ERA were 16.2%, 12.7%, and 6.8% and 14%, 8%, and 2% faster. For the larger workload l_5, AMS-ERA was 11.5%, 4.5%, and 0.2% faster. For larger workloads, both AMS-ERA and IDaPs exhibit similar performance.

We note that as the workload increases, the comparative performance of AMS-ERA against the compared schedulers for execution runtime also decreases. It can be asserted that this is due to the large number of disk IO reads and writes required by the wordcount algorithm. As the frugal cluster consists of SD Cards with slower read/write speeds, it is imperative that the runtime be affected by the available hardware speeds.

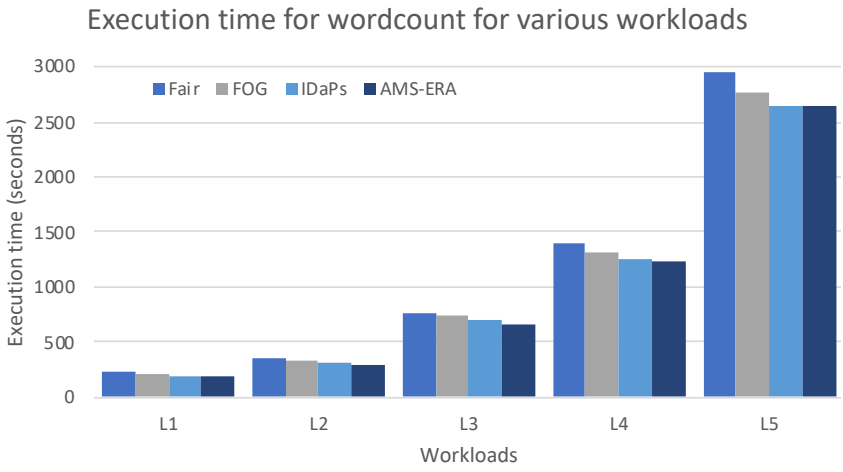
Our observation is confirmed when we compare the wordcount workload execution runtimes with the terasort execution runtimes. As mentioned earlier, terasort requires much less IO-intensive read/write operations to the disk; therefore, the expected runtime would be lower. For smaller workloads l_1 and l_2, the execution runtimes of terasort for the proposed AMS-ERA are 38.4%, 25.9%, and 20.5% and 34.9%, 20.7%, and 17.8% faster compared to Fair, FOG, and IDaPs, respectively.

For workloads l_3 and l_4, with a total of 64 and 128 map jobs, the execution times for AMS-ERA were 31%, 18%, and 13% and 26%, 14.4%, and 11% faster. For the larger workload l_5, AMS-ERA was 18.7%, 12.1%, and 7.8% faster.

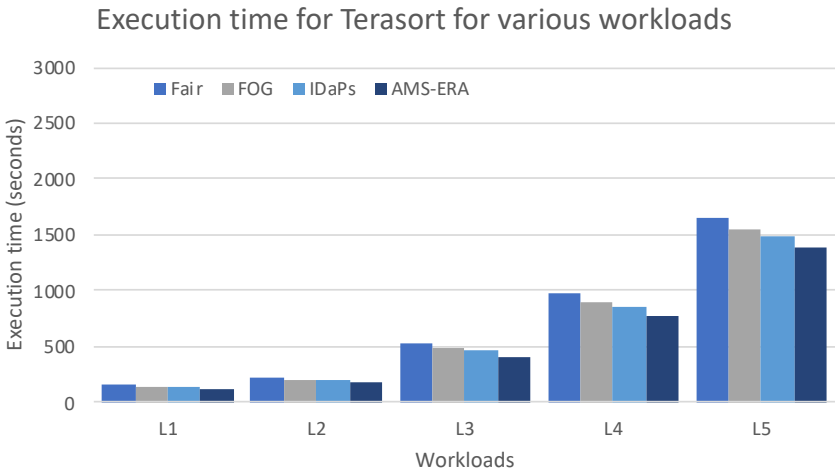
Figure 5b shows the comparison of the execution time of the five terasort workloads. As terasort is a comparatively less disk IO-intensive application, the AMS-ERA compares well with Fair, FOG, and IDaPs for all ranges of workloads.

From these results, we observe that in the worst-case scenario for workload l_5, where the entire dataset is required for execution (approx. 20 GB), both AMS-ERA and IDaPS exhibit similar performance. For smaller datasets and workloads, the proposed AMS-ERA performs significantly better. Figure 6a,b show the comparison of AMS-ERA performance

percentage against the compared schedulers for the various workflows using wordcount and terasort benchmarks.

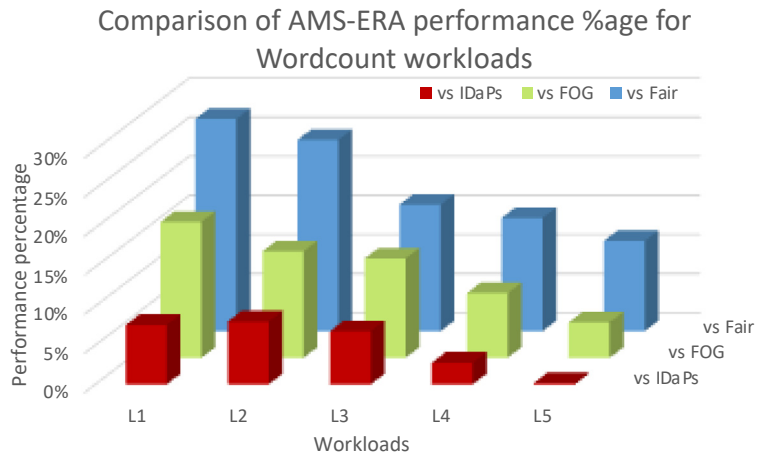


(**a**)



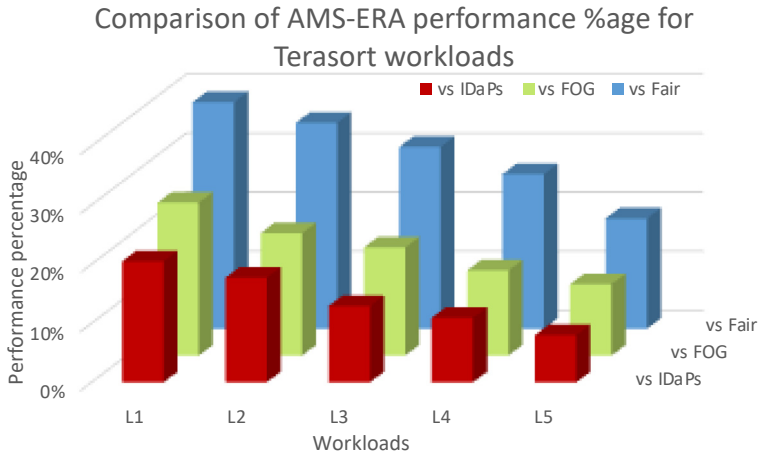
(**b**)

Figure 5. (**a**) A comparison of the execution times in seconds for wordcount jobs with workloads $\{l_1, l_2, l_3, l_4, l_5\}$ between Hadoop-Fair, FOG, IDaPS, and AMS-ERA. (**b**) depicts the execution times for terasort with the same workloads.



(**a**)

Figure 6. *Cont.*



(**b**)

Figure 6. (**a**) A comparison of the performance percentage of AMS-ERA execution times against Hadoop-Fair, FOG, and IDaPS for wordcount jobs with workloads $\{l_1, l_2, l_3, l_4, l_5\}$ (**b**) Performance percentage of AMS-ERA against Hadoop-Fair, FOG, and IDaPS for terasort.

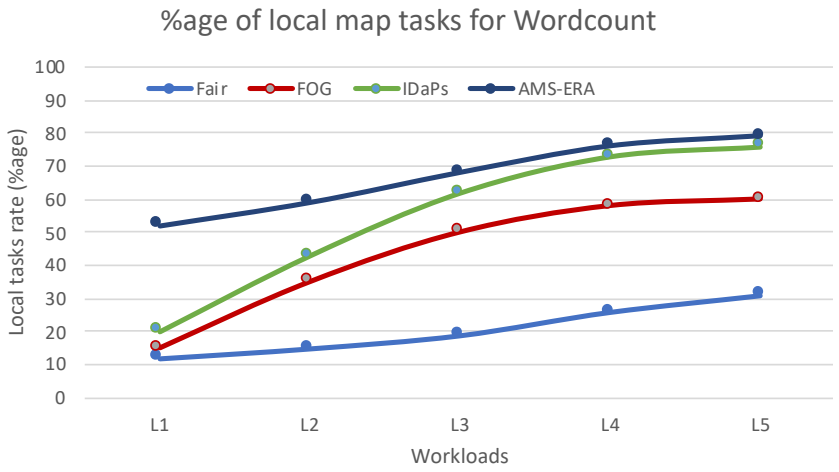
4.5. Local Job Placement and Resource Utilization

The default Hadoop Fair scheduling scheme distributes data without considering the computing capacity of nodes or network delay, resulting in poor performance. This lack of optimization leads to a higher percentage of non-local task executions and data transfer overhead compared to alternative schemes. Moreover, it overlooks the heterogeneity of the available nodes in the cluster. Consequently, the failure to account for these differences results in the suboptimal placement of map tasks in the cluster, thereby leading to poor performance.

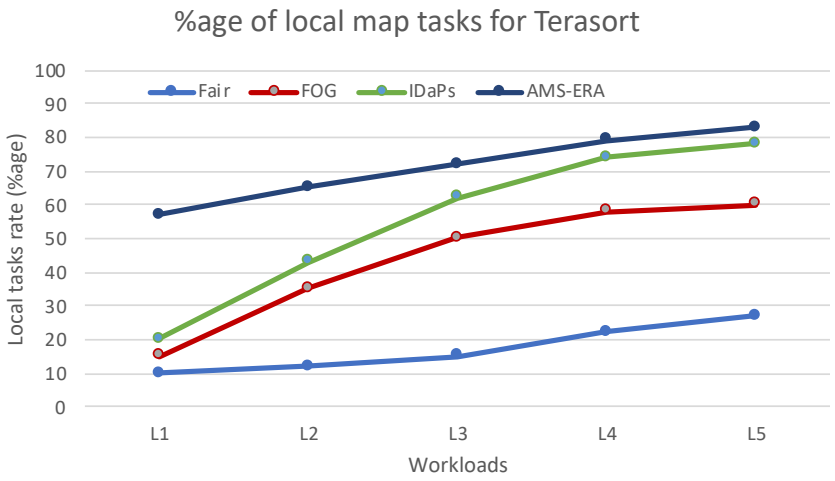
The proposed AMS-ERA assigns resource-intensive jobs to high-performance nodes within each group, sorting nodes in descending order based on their capacity range, including CPU, mem, and disk. Additionally, with higher cluster utilization, more jobs complete their execution quickly, enabling YARN to release resources sooner. Less demanding jobs are allocated to nodes that best match their resource requirements. Consequently, the system minimizes resource wastage and improves load balancing both between and within groups of heterogeneous nodes.

Figure 7a shows the percentage of locally assigned map tasks. All three schedulers outperform the Hadoop fair scheduler. For the wordcount workloads, as the number of map tasks increases, the utilization of resources for the proposed AMS-ERA also improves. For smaller workloads l_1, AMS-ERA outperforms the comparison with 52% local placement compared to 20% for IDaPS and 14% for FOG. With larger workloads l_5, the locality improves up to 79% for AMS-ERA compared to 76% for IDaPS and 61% for FOG. We observe a similar task placement rate (percentage) for terasort workloads as can be seen in Figure 7b. This shows that the AMS-ERA optimizes task locality based on resource availability in the cluster.

Figure 8a shows a comparison of the percentage of resource utilization for wordcount workload l_3. The AMS-ERA utilizes the highest resources effectively to complete the workload at the earliest. This shows the AMS-ERA job placement in the cluster effectively outperforms the Hadoop fair, FOG, and IDaPS. Figure 8b shows similar results for a terasort workload. As terasort is not disk-intensive, we can see that AMS-ERA has the highest average CPU and memory utilization; however, the disk utilization is slightly lower. Given these results, we can assume that AMS-ERA successfully considered the availability of resources in the cluster when placing the jobs. As a consequence of lower disk utilization, the map tasks for terasort were placed in high-performing nodes such as w6 and w9, which resulted in faster execution times.

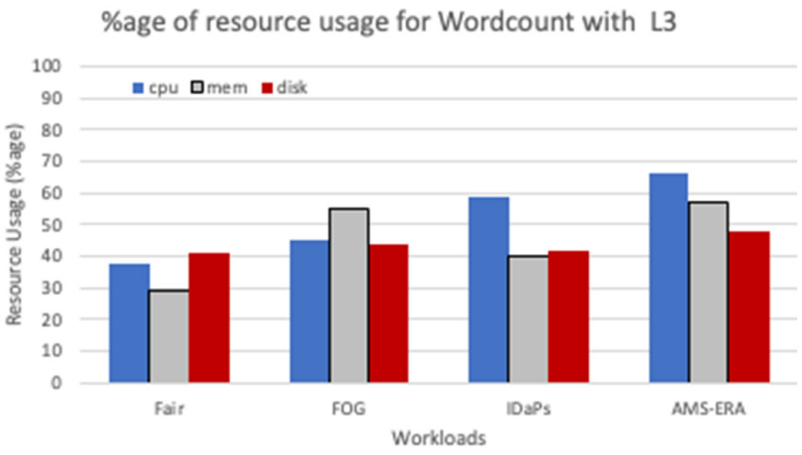


(**a**)



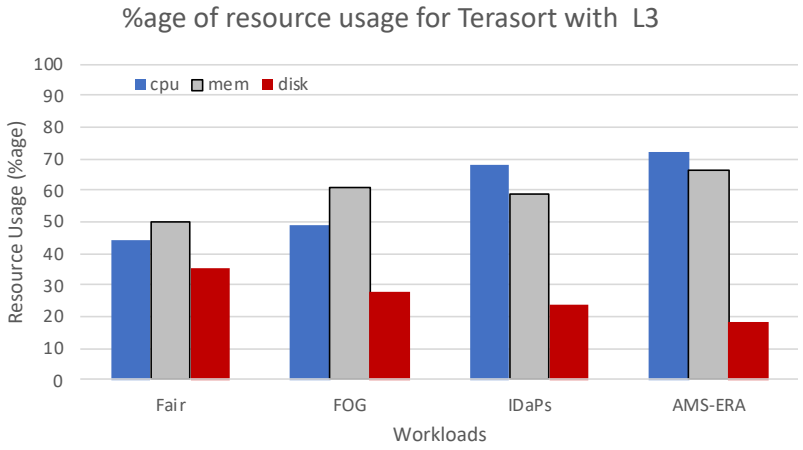
(**b**)

Figure 7. A comparison local task allocation rate (percentage) for AMS-ERA, Hadoop-Fair, FOG, IDaPS for (**a**) wordcount jobs with workloads $\{l_1, l_2, l_3, l_4, l_5\}$ and (**b**) for terasort jobs.



(**a**)

Figure 8. *Cont.*



(**b**)

Figure 8. A comparison of the percentage of CPU, mem, and disk resource utilization for (**a**) a wordcount workload l_3 and (**b**) a terasort workload.

4.6. Cost of Frugal Hadoop Cluster Setup

Building a Hadoop cluster with a diverse range of SBC models, each offering varying CPU, memory, and storage resources, allowed for diversification. This approach facilitates cost optimization by selecting models based on their price–performance ratio and the specific demands of the workload. The cost of our cluster setup was USD 822 for the 11 SBC devices along with networking essentials (cables, 2× Gigabit Switches, a router) and SD Card storage media.

During our experimental investigations, we observed a notable performance gap between the previous-generation RPi 3B nodes and traditional PC setups, with the former exhibiting suboptimal performance levels. However, with the introduction of AMS-ERA, which accounts for the heterogeneous nature of resources within the cluster, we observed significant improvements in execution times. Looking forward, we anticipate even greater performance enhancements with the latest RPi 5 nodes, which boast improved onboard resources compared to their predecessors.

This evolution in hardware capabilities underscores the potential for frugal SBC-based edge devices to not only enhance performance but also contribute to sustainability and cost-effectiveness in data processing applications. With the anticipated decrease in the cost of RPi 5 devices and their promising performance metrics, they present a compelling option for achieving both sustainability and cost-effectiveness in edge computing environments.

5. Conclusions and Future Work

In this work, we proposed Adaptive Multi-criteria Selection for Efficient Resource Allocation (AMS-ERA) in Frugal Heterogeneous Hadoop Clusters, addressing the critical challenge of resource allocation in clusters with frugal Single-Board Computers (SBCs). By considering the CPU, memory, and disk requirements for jobs and aligning them with available resources in the cluster, AMS-ERA optimizes resource allocation for optimal performance. Through K-means clustering, available resources are profiled and ranked based on similarity and proximity, enabling dynamic job placement. A dAHP refines the selection process by integrating historical data through Hadoop APIs. Jobs are then assigned to the most suitable nodes, ensuring load balancing in the heterogeneous cluster. Compared to Hadoop-Fair, FOG, and IDaPS scheduling strategies, AMS-ERA demonstrates superior performance, reducing execution time by 27.2%, 17.4%, and 7.6%, respectively, in terasort and wordcount benchmarks. The results show that the AMS-ERA is robust and performs consistently well across the diversified Map Reduce-based applications with various workload sizes. Furthermore, these also demonstrate that AMS-ERA ensures reduced execution time and improved data locality compared to Hadoop Fair, FOG, and

IDaPS. This study underscores the effectiveness of AMS-ERA in optimizing data layout, maximizing parallelism, and accommodating resource constraints in frugal SBC-based Hadoop clusters, paving the way for enhanced big data processing performance in resource-constrained environments.

AMS-ERA introduces a dynamic and adaptive approach to resource allocation, which could revolutionize how operational tasks are managed in Hadoop clusters. By profiling and ranking available resources and then aligning them with the job requirements, operational practices would become more efficient and responsive to workload demands. The capability to profile resources using K-means clustering and assign jobs based on a dAHP provides a flexible mechanism for job scheduling. This flexibility can lead to a more balanced workload, reducing bottlenecks, and potentially allowing operations teams to focus on other critical aspects of cluster management.

Since AMS-ERA is designed for frugal clusters with SBCs, its adaptive resource allocation mechanism could significantly impact edge computing. It could allow edge devices to participate in larger Hadoop clusters more effectively, opening new possibilities for data processing closer to the data source. The AMS-ERA framework could facilitate the deployment of Hadoop clusters in more constrained environments, like IoT applications or remote sites with limited infrastructure. By optimizing resource allocation and reducing execution time, AMS-ERA can indirectly lead to reduced energy consumption and operational costs. This is particularly relevant in SBC-based clusters, where energy efficiency is crucial.

At the moment, AMS-ERA is limited to CPU, memory, and disk utilization when considering job placement in the cluster. This scope of criteria might not cover all aspects of resource allocation efficiency, such as network bandwidth or I/O throughput. While the current version of AMS-ERA does not explicitly incorporate these factors, they are indirectly addressed through load balancing and job placement. Although AMS-ERA uses a dAHP to adapt to changes, there could be limitations in handling extreme fluctuations or sudden spikes in demand. This may lead to suboptimal resource utilization or load balancing in some scenarios. In the future, we intend to extend it to consider network localization, network capacities, and node/rack-based job placement. Furthermore, we intend to test AMS-ERA integrating real-time workflow datasets, enabling more robust and efficient performance evaluations. This could enhance its application in environments where real-time data processing is critical, such as stream processing or online analytics.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The author would like to acknowledge the support of Prince Sultan University for the payment of the article processing charges.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Awaysheh, F.M.; Tommasini, R.; Awad, A. Big Data Analytics from the Rich Cloud to the Frugal Edge. In Proceedings of the 2023 IEEE International Conference on Edge Computing and Communications (EDGE), Chicago, IL, USA, 2–8 July 2023; pp. 319–329.
2. Qin, W. How to Unleash Frugal Innovation through Internet of Things and Artificial Intelligence: Moderating Role of Entrepreneurial Knowledge and Future Challenges. *Technol. Forecast. Soc. Chang.* **2024**, *202*, 123286. [CrossRef]
3. Neto, A.J.A.; Neto, J.A.C.; Moreno, E.D. The Development of a Low-Cost Big Data Cluster Using Apache Hadoop and Raspberry Pi. A Complete Guide. *Comput. Electr. Eng.* **2022**, *104*, 108403. [CrossRef]
4. Vanderbauwhede, W. Frugal Computing—On the Need for Low-Carbon and Sustainable Computing and the Path towards Zero-Carbon Computing. *arXiv* **2023**, arXiv:2303.06642.
5. Chandramouli, H.; Shwetha, K.S. Integrated Data, Task and Resource Management to Speed Up Processing Small Files in Hadoop Cluster. *Int. J. Intell. Eng. Syst.* **2024**, *17*, 572–584. [CrossRef]
6. Han, T.; Yu, W. A Review of Hadoop Resource Scheduling Research. In Proceedings of the 2023 8th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), Okinawa, Japan, 23–25 November 2023; pp. 26–30.
7. Jeyaraj, R.; Paul, A. Optimizing MapReduce Task Scheduling on Virtualized Heterogeneous Environments Using Ant Colony Optimization. *IEEE Access* **2022**, *10*, 55842–55855. [CrossRef]

8. Saba, T.; Rehman, A.; Haseeb, K.; Alam, T.; Jeon, G. Cloud-Edge Load Balancing Distributed Protocol for IoE Services Using Swarm Intelligence. *Clust. Comput.* **2023**, *26*, 2921–2931. [CrossRef]
9. Guo, Z.; Fox, G. Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, Canada, 13–16 May 2012; pp. 714–716.
10. Bae, M.; Yeo, S.; Park, G.; Oh, S. Novel Data-placement Scheme for Improving the Data Locality of Hadoop in Heterogeneous Environments. *Concurr. Comput.* **2021**, *33*, e5752. [CrossRef]
11. Bawankule, K.L.; Dewang, R.K.; Singh, A.K. Historical Data Based Approach for Straggler Avoidance in a Heterogeneous Hadoop Cluster. *J. Ambient Intell. Humaniz. Comput.* **2021**, *12*, 9573–9589. [CrossRef]
12. Thakkar, H.K.; Sahoo, P.K.; Veeravalli, B. RENDA: Resource and Network Aware Data Placement Algorithm for Periodic Workloads in Cloud. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 2906–2920. [CrossRef]
13. Ghazali, R.; Adabi, S.; Rezaee, A.; Down, D.G.; Movaghar, A. CLQLMRS: Improving Cache Locality in MapReduce Job Scheduling Using Q-Learning. *J. Cloud Comput.* **2022**, *11*, 45. [CrossRef]
14. Ding, F.; Ma, M. Data Locality-Aware and QoS-Aware Dynamic Cloud Workflow Scheduling in Hadoop for Heterogeneous Environment. *Int. J. Web Grid Serv.* **2023**, *19*, 113–135. [CrossRef]
15. Postoaca, A.-V.; Negru, C.; Pop, F. Deadline-Aware Scheduling in Cloud-Fog-Edge Systems. In Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, Australia, 11–14 May 2020; pp. 691–698.
16. Vengadeswaran, S.; Balasundaram, S.R.; Dhavakumar, P. IDaPS—Improved Data-Locality Aware Data Placement Strategy Based on Markov Clustering to Enhance MapReduce Performance on Hadoop. *J. King Saud Univ. Comput. Inf. Sci.* **2024**, *36*, 101973. [CrossRef]
17. Adnan, A.; Tahir, Z.; Asis, M.A. Performance Evaluation of Single Board Computer for Hadoop Distributed File System (HDFS). In Proceedings of the 2019 International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 24–25 July 2019; pp. 624–627.
18. Qureshi, B.; Koubaa, A. On Energy Efficiency and Performance Evaluation of Single Board Computer Based Clusters: A Hadoop Case Study. *Electronics* **2019**, *8*, 182. [CrossRef]
19. Fati, S.M.; Jaradat, A.K.; Abunadi, I.; Mohammed, A.S. Modelling Virtual Machine Workload in Heterogeneous Cloud Computing Platforms. *J. Inf. Technol. Res.* **2020**, *13*, 156–170. [CrossRef]
20. Sebbio, S.; Morabito, G.; Catalfamo, A.; Carnevale, L.; Fazio, M. Federated Learning on Raspberry Pi 4: A Comprehensive Power Consumption Analysis. In Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing, Taormina, Italy, 4–7 December 2023; ACM: New York, NY, USA, 2023; pp. 1–6.
21. Shwe, T.; Aritsugi, M. Optimizing Data Processing: A Comparative Study of Big Data Platforms in Edge, Fog, and Cloud Layers. *Appl. Sci.* **2024**, *14*, 452. [CrossRef]
22. Raspberry Pi. Available online: https://www.raspberrypi.com/ (accessed on 7 May 2024).
23. Lee, E.; Oh, H.; Park, D. Big Data Processing on Single Board Computer Clusters: Exploring Challenges and Possibilities. *IEEE Access* **2021**, *9*, 142551–142565. [CrossRef]
24. Lambropoulos, G.; Mitropoulos, S.; Douligeris, C.; Maglaras, L. Implementing Virtualization on Single-Board Computers: A Case Study on Edge Computing. *Computers* **2024**, *13*, 54. [CrossRef]
25. Mills, J.; Hu, J.; Min, G. Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT. *IEEE Internet Things J.* **2020**, *7*, 5986–5994. [CrossRef]
26. Krpic, Z.; Loina, L.; Galba, T. Evaluating Performance of SBC Clusters for HPC Workloads. In Proceedings of the 2022 International Conference on Smart Systems and Technologies (SST), Osijek, Croatia, 19–21 October 2022; pp. 173–178.
27. Lim, S.; Park, D. Improving Hadoop Mapreduce Performance on Heterogeneous Single Board Computer Clusters. *SSRN Preprint* **2023**. [CrossRef]
28. Srinivasan, K.; Chang, C.Y.; Huang, C.H.; Chang, M.H.; Sharma, A.; Ankur, A. An Efficient Implementation of Mobile Raspberry Pi Hadoop Clusters for Robust and Augmented Computing Performance. *J. Inf. Process. Syst.* **2018**, *14*, 989–1009. [CrossRef]
29. Fu, W.; Wang, L. Load Balancing Algorithms for Hadoop Cluster in Unbalanced Environment. *Comput. Intell. Neurosci.* **2022**, *2022*, 1545024. [CrossRef]
30. Yao, Y.; Gao, H.; Wang, J.; Sheng, B.; Mi, N. New Scheduling Algorithms for Improving Performance and Resource Utilization in Hadoop YARN Clusters. *IEEE Trans. Cloud Comput.* **2021**, *9*, 1158–1171. [CrossRef]
31. Javanmardi, A.K.; Yaghoubyan, S.H.; Bagherifard, K.; Nejatian, S.; Parvin, H. A Unit-Based, Cost-Efficient Scheduler for Heterogeneous Hadoop Systems. *J. Supercomput.* **2021**, *77*, 1–22. [CrossRef]
32. Ullah, I.; Khan, M.S.; Amir, M.; Kim, J.; Kim, S.M. LSTPD: Least Slack Time-Based Preemptive Deadline Constraint Scheduler for Hadoop Clusters. *IEEE Access* **2020**, *8*, 111751–111762. [CrossRef]
33. Zhou, R.; Li, Z.; Wu, C. An Efficient Online Placement Scheme for Cloud Container Clusters. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 1046–1058. [CrossRef]
34. Zhou, Z.; Shojafar, M.; Alazab, M.; Abawajy, J.; Li, F. AFED-EF: An Energy-Efficient VM Allocation Algorithm for IoT Applications in a Cloud Data Center. *IEEE Trans. Green Commun. Netw.* **2021**, *5*, 658–669. [CrossRef]

35. Zhou, Z.; Abawajy, J.; Chowdhury, M.; Hu, Z.; Li, K.; Cheng, H.; Alelaiwi, A.A.; Li, F. Minimizing SLA Violation and Power Consumption in Cloud Data Centers Using Adaptive Energy-Aware Algorithms. *Future Gener. Comput. Syst.* **2018**, *86*, 836–850. [CrossRef]
36. Banerjee, P.; Roy, S.; Sinha, A.; Hassan, M.; Burje, S.; Agrawal, A.; Bairagi, A.K.; Alshathri, S.; El-Shafai, W. MTD-DHJS: Makespan-Optimized Task Scheduling Algorithm for Cloud Computing With Dynamic Computational Time Prediction. *IEEE Access* **2023**, *11*, 105578–105618. [CrossRef]
37. Zhang, L. Research on K-Means Clustering Algorithm Based on MapReduce Distributed Programming Framework. *Procedia Comput. Sci.* **2023**, *228*, 262–270. [CrossRef]
38. Postoaca, A.V.; Pop, F.; Prodan, R. H-Fair: Asymptotic Scheduling of Heavy Workloads in Heterogeneous Data Centers. 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. 366–369.
39. Guo, T.; Bahsoon, R.; Chen, T.; Elhabbash, A.; Samreen, F.; Elkhatib, Y. Cloud Instance Selection Using Parallel K-Means and AHP. In Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion, Auckland, New Zealand, 2–5 December 2019; ACM: New York, NY, USA, 2019; pp. 71–76.
40. Odroid Xu4. Available online: https://www.hardkernel.com/shop/odroid-xu4-special-price/ (accessed on 7 May 2024).
41. RockPro64. Available online: https://pine64.com/product/rockpro64-4gb-single-board-computer/ (accessed on 7 May 2024).
42. Herodotou, H.; Lim, H.; Luo, G.; Borisov, N.; Dong, L.; Cetin, F.; Babu, S. Starfish: A Self-Tuning System for Big Data Analytics. In Proceedings of the CIDR 2011—5th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, 9–12 January 2011; Conference Proceedings; pp. 261–272.
43. Syakur, M.A.; Khotimah, B.K.; Rochman, E.M.S.; Satoto, B.D. Integration K-Means Clustering Method and Elbow Method For Identification of The Best Customer Profile Cluster. *IOP Conf. Ser. Mater. Sci. Eng.* **2018**, *336*, 012017. [CrossRef]
44. Kim, H.-J.; Baek, J.-W.; Chung, K. Associative Knowledge Graph Using Fuzzy Clustering and Min-Max Normalization in Video Contents. *IEEE Access* **2021**, *9*, 74802–74816. [CrossRef]
45. Singh, A.; Das, A.; Bera, U.K.; Lee, G.M. Prediction of Transportation Costs Using Trapezoidal Neutrosophic Fuzzy Analytic Hierarchy Process and Artificial Neural Networks. *IEEE Access* **2021**, *9*, 103497–103512. [CrossRef]