computation

MDPI

Article

Optimizing Hadoop Scheduling in Single-Board-Computer-Based Heterogeneous Clusters

Basit Qureshi

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

Abstract: Single-board computers (SBCs) are emerging as an efficient and economical solution for fog and edge computing, providing localized big data processing with lower energy consumption. Newer and faster SBCs deliver improved performance while still maintaining a compact form factor and cost-effectiveness. In recent times, researchers have addressed scheduling issues in Hadoop-based SBC clusters. Despite their potential, traditional Hadoop configurations struggle to optimize performance in heterogeneous SBC clusters due to disparities in computing resources. Consequently, we propose modifications to the scheduling mechanism to address these challenges. In this paper, we leverage the use of node labels introduced in Hadoop 3+ and define a Frugality Index that categorizes and labels SBC nodes based on their physical capabilities, such as CPU, memory, disk space, etc. Next, an adaptive configuration policy modifies the native fair scheduling policy by dynamically adjusting resource allocation in response to workload and cluster conditions. Furthermore, the proposed frugal configuration policy considers prioritizing the reduced tasks based on the Frugality Index to maximize parallelism. To evaluate our proposal, we construct a 13-node SBC cluster and conduct empirical evaluation using the Hadoop CPU and IO intensive microbenchmarks. The results demonstrate significant performance improvements compared to native Hadoop FIFO and capacity schedulers, with execution times 56% and 22% faster than the best_cap and best_fifo scenarios. Our findings underscore the effectiveness of our approach in managing the heterogeneous nature of SBC clusters and optimizing performance across various hardware configurations.

Keywords: single-board computers; frugal edge computing; Hadoop; YARN; heterogeneous

Citation: Qureshi, B. Optimizing Hadoop Scheduling in Single-Board-Computer-Based Heterogeneous Clusters. *Computation* **2024**, *12*, 96. https://doi.org/10.3390/computation12050096

Academic Editor: Xiaoqiang Hua

Received: 2 April 2024
Revised: 5 May 2024
Accepted: 7 May 2024
Published: 9 May 2024

1. Introduction

Low-cost single-board computers (SBCs) are emerging as a smart choice for fog and edge computing, bringing innovation and sustainability to the forefront. These small devices offer benefits that go beyond traditional computing, such as reducing energy use and enabling localized data processing [1]. Unlike traditional computing setups that often require substantial power consumption, SBCs are designed to operate efficiently with minimal energy usage. This inherent characteristic makes them ideal candidates for powering edge devices, which are frequently deployed in remote or off-grid locations where energy resources may be limited. By consuming less energy, SBC-based edge devices contribute to energy conservation and help lower carbon emissions, easing the burden on the environment [2]. From reducing energy consumption to enabling localized processing, the integration of SBCs in edge devices holds significant promise for mitigating environmental impact while fostering sustainable technological advancements.

Over the past decade, Apache Hadoop has become a leading framework for big data processing [3]. The Hadoop framework has established itself as a key player, allowing distributed computing across a wide range of nodes. In their work [4], the researchers built a Hadoop cluster using Raspberry Pi to process images, working with datasets of various sizes. They then measured how long it takes this SBC-based cluster to complete tasks compared to a traditional PC. The findings suggest that for smaller datasets, the Raspberry Pi cluster finishes the tasks in less time than the PC. In their study [5], Neto

and colleagues detail the creation, testing, and monitoring of a budget-friendly big data cluster built with Raspberry Pi 4B devices running Apache Hadoop. The results indicate that this combination of Raspberry Pi and Apache Hadoop provides a reliable and cost-efficient solution for low-cost big data clusters. The authors in [6] explore the application of Apache Hadoop on a cluster of Raspberry Pi 4B single-board computers (SBCs) to evaluate their viability as low-cost, energy-efficient platforms for big data processing. By conducting various Hadoop benchmarks such as WordCount, Terasort, DFSIO, etc., and experimenting with different storage configurations, the study shows that Raspberry Pi clusters can successfully manage large workloads, providing a practical alternative to traditional server setups. In their work [7], Lambropoulos et al. investigate the use of single-board computers (SBCs) such as Raspberry Pi 4B for edge computing, showcasing a successful shift from conventional x86-based infrastructure to SBC-based clusters. Although this setup experiences increased CPU usage and storage latency, it brings significant energy savings, using just one-ninth of the power compared to traditional systems. The authors address hardware compatibility issues and storage performance challenges, proposing future research into dedicated storage solutions and enhanced hardware customization to mitigate the limitations in edge computing environments.

However, its traditional scheduling mechanisms, designed for homogeneous environments, often fall short when deployed in clusters with heterogeneous resources, such as single-board computers (SBCs) [8,9]. Hadoop does not inherently account for the varying capacities of nodes and heterogeneity in a cluster, leading to issues like overloading frugal SBCs with intensive tasks or allocating insufficient memory, resulting in task failures [2,6,10]. This is particularly challenging in resource-frugal clusters where individual nodes, like Raspberry Pi or Odroid devices, have limited CPU processing power and memory capacity. Such constraints can cause performance bottlenecks, impacting the efficiency and scalability of Hadoop clusters.

To address these limitations, this study proposes a novel scheduling mechanism that introduces a *Frugality Index* (Findex) and an adaptive configuration (adaptiveConfig) policy to enhance resource allocation and system efficiency. The Findex categorizes SBC nodes based on their processing capabilities and memory size, allowing for smarter task distribution. Additionally, the adaptive configuration policy adapts resource allocation in real-time, based on workload and cluster status. This dynamic approach contrasts with traditional scheduling policies in Hadoop, which often lack flexibility and lead to inefficiencies in heterogeneous clusters. By customizing container placement and resource allocation, this new scheduling mechanism aims to improve the performance of frugal SBC-based clusters, offering faster execution times while consuming less power. The proposed mechanism leverages Hadoop's node-labeling feature [11], allowing administrators to categorize nodes with specific labels, such as "Hi", "Med", or "Low", based on their capabilities. This approach, coupled with the adaptive configuration, ensures tasks are assigned to nodes with appropriate resources, minimizing job completion times and maximizing parallelism.

We construct a cluster composed of 13 SBC devices and conduct various experiments to test the proposed scheduling mechanism. The evaluation includes four fundamental scenarios that delineate task distribution patterns for various work loads of different sizes, namely best_capacity, worst_capactiy, best_fifo, and worst_fifo. By focusing on both CPU-intensive and I/O-intensive workloads, such as WordCount and Terasort, we extensively test the proposed for various workloads with two Hadoop InputSplit size settings. The experiments measure task completion times, CPU utilization, memory usage, and network traffic to compare the performance of native Hadoop configurations with the proposed adaptive scheduling mechanism. Results from these experiments demonstrate that the proposed scheduling mechanism, incorporating the Findex and adaptiveConfig policy, significantly outperforms traditional Hadoop configurations. This improvement is evident in the task completion times and resource utilization metrics, indicating that the new approach effectively addresses the challenges of task assignment in heterogeneous SBC-based clusters. By optimizing resource allocation and enhancing system flexibility, the

proposed scheduling mechanism offers a promising solution for improving the performance and sustainability of frugal SBC-based Hadoop clusters.

The rest of the paper is organized as follows. Section 2 presents relevant work and background. Section 3 details the proposed adaptive frugal configuration policy. Section 4 presents extensive performance evaluation of the SBC cluster followed by discussion and future directions in Section 5. Section 6 concludes this work.

2. Background and Related Works

In this section, we present SBC properties; Apache Hadoop YARN components and architecture; and the motivation to design scheduling policies in YARN for frugal SBC-based clusters. We also discuss recent works on improving YARN performance on heterogeneous clusters.

2.1. Single-Board Computers

SBCs are compact computing devices built on a single circuit board, encompassing all essential components such as CPU, memory, storage, and input/output interfaces. These boards offer a range of advantages, particularly in terms of a small form factor while being power and energy-efficient. Their compact design makes them suitable for applications where space is limited, and their integrated components contribute to lower power consumption compared to traditional desktop computers. Additionally, many SBCs are designed to operate efficiently on minimal power, making them ideal for battery-powered devices and scenarios where energy efficiency is paramount. SBCs also come with certain limitations. While they offer sufficient processing power for many tasks, their performance may be limited compared to desktop computers, particularly for demanding computational tasks such as big data applications. Despite these limitations, SBCs remain popular and versatile computing platforms used in various applications. Examples of well-known SBCs include the Raspberry Pi (RPi), Arduino, NVIDIA Jetson Nano, Odroid XU4, and BeagleBone Black. Each of these devices offers unique features and specifications, catering to a diverse range of use cases while embodying the principles of compactness, efficiency, and affordability that define the SBC ecosystem.

Table 1 provides a summary of SBCs used in this study. Raspberry Pi computers are by far the most popular SBC and are widely used in industrial, healthcare, robotics, and IoT applications. First released in 2012, they are cost-effective, energy efficient, and widely accessible and have been used in various studies. A major drawback with earlier generation RPis was the computational capacity, as highlighted in our earlier work in [12]. With newer models 3B+, 4B, and the fifth generation, the use of improved on-board processors has significantly improved the performance of individual SBCs. Additionally, the increased upgraded RAM module using LPDDR4X RAM available on RPI 4B and 5 is a useful upgrade. Gigabit Ethernet and HDMI come standard with these SBCs for faster connectivity and A/V displays. We also use Odriod XU-4 (Odroid Xu-4 https://www.odroid.co.uk/hardkernel-odroid-xu4/odroid-xu4, accessed on 8 May 2024) SBCs that use Samsung Exynos Octa core ARM processors with a 2 GHz quad-core Cortex-A15 and 1.3 GHz quad-core Cortex-A7 processor. The Xu-4 has 2 GB DDR3 RAM, Gigabit Ethernet, and a standard HDMI port. The Pine64 RockportPro64 (Pine 64 RockPro64 https://pine64.com/product/rockpro64-4gb-single-board-computer/, accessed on 8 May 2024) is another SBC used in this work. It is powered by a Rockchip RK3399 Hexa-Core (dual ARM Cortex A72 and quad ARM Cortex A53) 64-Bit Processor with MALI T-860 Quad-Core GPU. The ROCKPro64 is equipped with 4 GB LPDDR4 system memory and 128 Mb SPI boot Flash. All of these SBCs support microSD cards for storage with varying sizes, including 64 GB. Odroid Xu4 and Rockpro64 also support the faster eMMC modules.

Table 1. Specifications of various SBCs used in this work.

	Raspberry Pi 5	**Pine64 Rockpro64**	**Raspberry Pi 3B+**	**Odriod XU-4**
Processor	2.4 GHz quad-core 64-bit ARM Cortex A76	1.8 GHz Hexa Rockchip RK3399 ARM Cortex A72 and 1.4 GHz Quad Cortex-A53	1.4 GHz 64-bit quad-core ARM Cortex-A53	Exynos5 Octa ARM Cortex-A15 Quad 2 Ghz and Cortex-A7 Quad 1.3 GHz
Memory	8 GB LPDDR4X-SDRAM	4 GB LPDDR4-SDRAM	1 GB LPDDR3-SDRAM	2 GB DDR3
Ethernet	Gigabit Ethernet	Gigabit Ethernet	300 Mbit/s	Gigabit Ethernet
GPU	VideoCore VII 800 MHz	Mali-T860 GPU 700 MHz	VideoCore IV 400 MHz	Mali-T628 MP6 600 MHz
A/V	HDMI	HDMI	HDMI 1.3	HDMI
Price (USD)	80	79.99	35	53
Release	2023	2018	2018	2016
Power	1.3 W idle; 8.6 W max	3.1 W idle; 10.9 W max	1.9 W idle; 5.1 W max	2.1 W idle; 6.4 W max

2.2. Apache Hadoop YARN

The Hadoop ecosystem encompasses a suite of open-source projects and tools revolving around the core Hadoop framework. Hadoop, a distributed computing framework, facilitates the storage and processing of vast datasets across clusters of commodity hardware. Central to this ecosystem is the Hadoop MapReduce, providing a programming model for distributed data processing, while YARN manages resource allocation. Hadoop YARN scheduling is a critical component of the Hadoop ecosystem, tasked with efficiently managing resources across the cluster. In Hadoop, the NameNode serves as the central component of the Hadoop Distributed File System (HDFS), managing metadata about the file system namespace and block locations. It directs client read and write requests and oversees the storage of data across the cluster's worker nodes, called DataNodes. The Resource Manager (RM), running on the master node, manages resource allocation and job scheduling and monitors their execution. Together, the NameNode and RM facilitate efficient distributed storage and processing. A DataNode is a worker node responsible for storing data blocks and ensuring data replication and availability. It communicates with the NameNode to report block information and handles read and write requests. Conversely, the Node Manager (NM) is a per-node agent managing resources and executing tasks on worker nodes. It reports available resources to the RM, launches and monitors containers, and ensures the proper execution of tasks. The Application Master (AM) manages the execution of individual applications within the cluster, negotiating resources from the RM, coordinating task execution, and monitoring progress.

A container represents a unit of resource allocation. When a client submits a MapReduce job to the Hadoop cluster, the RM receives the request and designates a worker node to host the AM in a container for the job. The NM on the worker node is notified of the job, which coordinates with the AM to request the required number of containers. The NM allocates resources to containers and launches the required number of containers on the worker node. These containers host the actual MapReduce tasks or application code. In case containers fail, YARN provides fault tolerance by swiftly detecting node failures through NM, which reports to the RM through periodic heatbeat messages. Tasks affected by node failures are rescheduled on available nodes, and the containers' states are recovered to ensure uninterrupted progress. Figure 1 shows the various components of the YARN architecture and the service flow. The RM employs its scheduler to allocate resources based on availability and predefined policies. YARN supports various scheduling policies such as FIFO, Capacity, and Fair schedulers, each with distinct resource allocation and job

prioritization methods. It dynamically manages the allocation of containers based on the available resources and the requirements of applications running on the cluster.

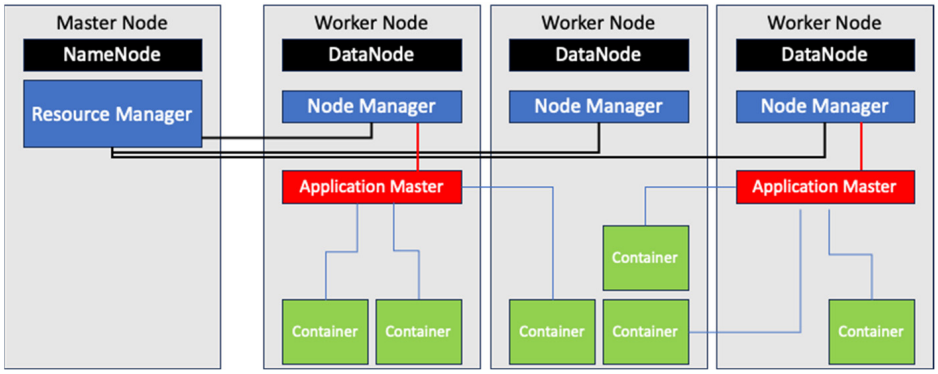


Figure 1. Hadoop YARN architecture and service flow.

Node labels [11] were introduced in Hadoop version 3. They allow administrators to categorize nodes based on specific attributes, such as hardware capabilities, geographical location, or other intended uses. This labeling system helps to achieve fine-grained control over where specific applications or workloads run. By directing specific applications to particular nodes, organizations can maintain better control over job distribution and system performance, contributing to overall efficiency and scalability in YARN-based environments.

2.3. Relevant Research Works

In this section, we present relevant research works focusing on Hadoop YARN performance in the context of heterogeneous clusters with limited onboard resources.

Jeyaraj et al., in [8], observe that 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. As a result, tasks may be assigned to nodes that are ill-suited for processing them efficiently, leading to resource contention and reduced performance. Bae, in [9], observed that in heterogeneous environments, Hadoop's subpar performance was mainly due to the equal block allocation across nodes in the cluster. The authors proposed a new data-placement scheme focusing on improving Hadoop's data locality by minimizing replicated data. This was achieved by modifying the scheduling policy to accommodate the selection and replication of the blocks with the highest likelihood of remote access.

V. Thesma et al., in [13], developed a low-cost distributed computing pipeline for cotton plant phenotyping using Raspberry Pi, Hadoop, and deep learning. They compare the performance of the Raspberry Pi-based Hadoop cluster in various configurations for high-throughput cotton phenotyping in field-based agriculture. Veerachamy, in [14], presents an agricultural irrigation recommendation and alert system using optimization and machine learning in Hadoop for sustainable agriculture. They use machine learning algorithms to forecast alerts based on various parameters such as air pressure, water level, humidity, etc.

Setiyawan, in [15], developed an Internet of Things (IoT)-based wireless engine diagnostic tool prototype using a Raspberry Pi. This plug-and-play tool is used for engine diagnostics in vehicle repair shops. In [6], researchers developed an intelligent personal assistant system based on IoT for people with disabilities. The proposed system utilizes Raspberry Pi as a control device for processing natural language input. Netinant et al., in [16], developed an IoT-driven smart home security and automation framework with voice commands. The proposed framework ensures the incorporation of components, including Raspberry Pi, relays, motion sensors, etc. The authors in [17] analyze the impact of lightweight mutual authentication for the healthcare IoT. The proposed technique significantly improves the disadvantages of IoT devices that lack computing power.

Recently, the researchers in [18] have directed their attention towards achieving energy-efficient remote data processing through the utilization of clusters comprised of single-board computers (SBCs), like Raspberry Pi, coupled with the Hadoop framework for handling large-scale data processing tasks in various contexts, including agriculture, smart cities, smart homes, healthcare, etc. Qureshi et al. in [10] developed a heterogenous cluster of 20 SBCs, including Raspberry Pis and Ordoid Xu-4s, for data analytics using Hadoop. They conducted various experiments to analyze the performance and energy efficiency of the cluster for workloads of various sizes. They observed that the performance of Raspberry Pi-based clusters was inferior to Ordoid Xu-4 machines due to the frugal nature of the devices. Lee, in [6], presents an in-depth investigation into Hadoop performance, focusing specifically on the latest generation Raspberry Pi cluster, built with RPi model 4B. They conduct a thorough examination of Apache Hadoop benchmarks and note that the cluster composed of the five latest models of SBCs can successfully process workloads of a few tera-bytes. Sooyoung L. et al., in [19], proposed modification to YARN to accommodate the heterogeneity due to the scalability in SBC-based clusters. They developed a master-driven and slave-driven policy that dynamically determines the onboard capacities of the nodes in the cluster. To evaluate their work, they construct a small cluster of Raspberry Pi computers and evaluate the performance of the proposed policies using Hadoop benchmarks. Neto et al., in [5], analyze the performance of Raspberry Pi-based clusters using various benchmarks, including *Terasort* and *DFSIO*. They note that clusters formed by Raspberry Pi have proved to be a viable and economical solution for carrying out tasks involving the use of big data. Nugroho et al. in [20] also design a parallel computing framework using Raspberry Pi clusters for IoT services and applications.

In this work, we leverage the use of YARN node labels to categorize the nodes in the SBC cluster based on their onboard properties. We also re-configure the default scheduling policies in YARN, such as the FIFO and Capacity schedulers, to improve task scheduling in the SBC cluster. We conduct extensive experimentation on a 13-node SBC cluster using Hadoop benchmarks to compare the performance of the proposed configuration policies. The next section details the proposed scheduling mechanism.

3. Proposed Scheduling Mechanism

3.1. Motivation and Limitations

In the Hadoop framework version 3+, node labels were introduced. Node labels are a powerful feature that enhances resource management and job scheduling flexibility in a YARN cluster. Node labels are essentially key-value pairs that allow administrators to categorize nodes based on their characteristics or intended use. This categorization is useful for creating logical partitions within the cluster, allowing workloads to be directed to specific sets of nodes based on their labels. It is pertinent that these characteristics of nodes within clusters on the edge made with resource-frugal devices would play a pivotal role in determining the performance of executing concurrent MapReduce tasks.

In [10], the authors observed that the performance of a Raspberry Pi 3B-based Hadoop cluster was inferior to Ordoid Xu-4 machines primarily due to the frugal nature of the onboard components on the devices. The RPi-based cluster in particular was more prone to failure due to lack of memory error frequently hindering the progress of MapReduce tasks. MapReduce tasks being dropped due to memory limits indicate issues such as inefficient memory usage within the application or insufficient memory resources allocated to the cluster. Upon further examination, it became apparent that a native Hadoop setup does not support concurrent execution of two or more map tasks on a node with only 1 GB of RAM. On the other hand, the Odriod Xu-4 SBC did not present similar performance bottlenecks due to memory restrictions. It was able to handle up to two containers per node/device. In earlier works [5,12,19], the researchers note that the native YARN settings do discern the limited capabilities of these devices. When the number of containers exceeds two on an SBC node, it can overwhelm the task queue within the scheduler. This overburdening of tasks can cause the system to become unresponsive, as it struggles to manage the concurrent

execution of tasks efficiently. Consequently, the system may reach a point where it becomes unresponsive to further requests or tasks, leading to a potential halt in job execution. In such scenarios, users may need to intervene by manually terminating the jobs to alleviate the strain on the system and restore its functionality. A Raspberry Pi Hadoop node equipped with 1 GB of RAM is unable to effectively carry out significant data processing tasks that necessitate simultaneous execution of multiple map tasks.

To this end, we modified the *mapreduce.map.memory.mb* property in the *mapred-site.xml* configuration file to maximize the memory limit to 852 MB. Table 2 shows the Hadoop and YARN configuration files. This limits the execution on the frugal RPi devices in the cluster to only one container, ensuring that the application does not crash. A similar observation is also made by the authors in [3], where the authors run into similar issues with regards to memory management. To alleviate this restriction, one approach is to increase the size of the swap partition on the host operating system to maximize the utilization of the virtual memory; however, this resulted in slower performance due to the significantly slow read/write speeds on the local storage media (SD cards). Regardless of these improvements, it is imperative that the physical memory constraint restricts parallelization within the cluster, effectively throttling the performance due to the frugal nature of the SBC devices.

Table 2. Hadoop YARN configuration properties used for resource-frugal SBC-based clusters.

Mapred-site.xml	**Value**
yarn.app.mapreduce.am.resource.mb	852
mapreduce.map.cpu.vcores	1
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	852
yarn.scheduler.maximum-allocation-vcores	8
yarn.nodemanager.vmem-pmem-ratio	2.1
yarn.node-labels.enabled	true
yarn.node-labels.fs-store.root-dir	hdfs://dir-path

In this section, we define node labels that categorize the onboard capacities of SBC nodes using a Findex. Using Hadoop node labels, the Findex is passed as a parameter to the RM, NM, and Application Manager to assign relevant containers to the frugal SBC node(s). We redefine YARN scheduling policies to adapt to the Findex and proposed an adaptiveConfig policy for scheduling jobs/tasks. The assignment of containers is prioritized and placed on frugal nodes within the cluster based on these parameters. This approach ensures efficient resource utilization and improves the system's overall efficiency by adaptively assigning MapReduce tasks according to the heterogeneous capacities of nodes within the SBC-based cluster.

3.2. Frugality Index and Node Labels

As mentioned earlier in the motivation, the node labeling in Hadoop allows administrators to categorize and assign nodes with specific labels based on their physical properties, such as CPU capacity, memory size, or disk space. This technique can help Hadoop's YARN framework to allocate tasks more intelligently, ensuring that each task is assigned to a node that has the appropriate resources to handle it effectively. This strategy is particularly useful in heterogeneous clusters [21], where nodes may vary significantly in terms of performance and capability.

To enable the node labels in YARN, we modify the *yarn.node-labels.enabled=true* property in *yarn-site.xml*. Furthermore, we define the values of the node labels using a custom script. These values are "Hi", "Med", and "Low", categorizing the nodes based on their onboard available resources. Further details can be seen in Table 3. The NM on each node in the cluster determines the Findex based on the local device/node's physical characteristics. The Findex value implicitly is derived from the size of the on-board memory available on the device. The Findex value is communicated from the NM to the RM along with the heartbeat messages. This is to reduce the overall communication overhead. The RM considers the updates along with the scheduling policies to place containers on the various worker nodes.

Table 3. Frugality Index guideline.

Findex	Device	CPU	pCores	Memory	Node-Label
4	Raspberry Pi 5	2.4 GHz	4	8 GB	Hi
3	Raspberry Pi 4	1.5 GHz	4	4 GB	Hi
2	Pine64 Rockpro64	1.8 GHz	6	4 GB	Med
2	Odroid Xu4	2.0 GHz	8	2 GB	Med
1	Raspberry Pi 3B	1.4 GHz	4	1 GB	Low
1	Raspberry Pi 2	900 MHz	4	1 GB	Low

3.3. Re-Configuring YARN Heartbeat Messages

The RM in Hadoop YARN determines the resources required for a job based on the application's resource requests, the cluster's available resources, and any configured scheduling policies. When a user submits a job to the RM, the application specifies its resource requirements, including CPU cores, memory, and other resources through the Application Manager. When the AM initiates, it posts a request to the scheduler. Based on the provided parameters, the scheduler requests *ResourceTracker* to launch the AM. It finds a suitable *datanode* that supports the AM container and assigns it to the application. The Application Manager launches the AM on the worker node. A *datanode* executes the NM. The NM periodically updates the RM to inform about their available resources through a process called the *heartbeat* mechanism. The NM periodically sends *heartbeat* messages to the RM to indicate their availability and resource status. These *heartbeat* messages contain information such as the node's total memory, CPU cores, available memory, available CPU cores, and other resource metrics. The frequency of the heartbeat message can be provided in the YARN configuration using the property *yarn.nodemanager.node-labels.resync-interval-ms*. We have set its value to 1 min.

The RM receives these *heartbeat* messages from all active NMs in the cluster. Based on its resource allocation decision, the RM communicates with specific NMs to allocate containers for executing tasks. Each container is launched with the specified resource allocation, and tasks within the containers begin execution. Throughout the job's execution, NMs continue to send periodic heartbeat messages to the RM, providing updates on container status and resource usage. Figure 2 illustrates the information flow between various components of the RM and NMs. The Findex values are used by the scheduling mechanism to determine appropriate resources for containers and assign tasks to frugal nodes for computation in the cluster.

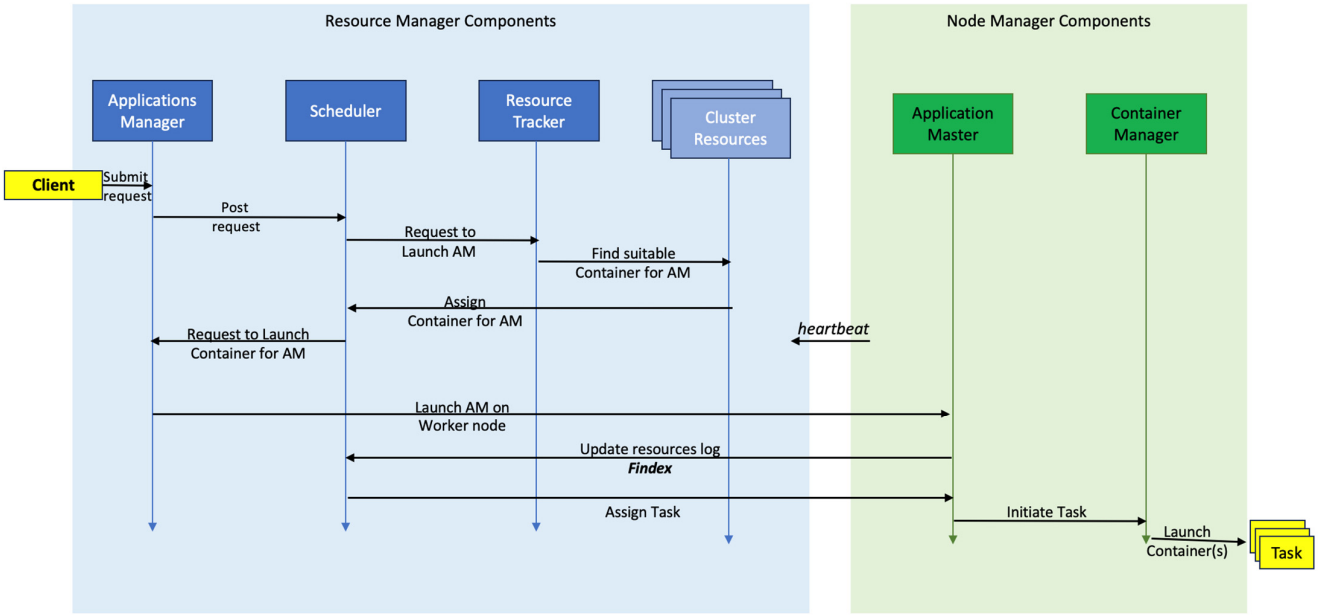


Figure 2. Information flow between various components of RM and NM in YARN.

3.4. Adaptive Fair Scheduling Scheme

Native YARN offers three distinct scheduling policies: FIFO, Capacity, and Fair. The FIFO scheduling policy, being the simplest, executes applications in the order of their arrival, without permitting concurrent execution. Consequently, long-running applications have the potential to block the execution of shorter jobs that may only require a fraction of the available resources. The Capacity scheduling policy enables the definition of multiple queues each assigned with a percentage of cluster resources. Each queue is assured a minimum resource allocation, facilitating concurrent execution of applications submitted to different queues. In addition, applications within the same queue may also run concurrently, subject to the queue policy. The Fair scheduling policy, similar to the Capacity policy, features queues with minimum resource guarantees. However, instead of statically partitioning resources, they are dynamically balanced among submitted jobs. These scheduling policies are set in the Hadoop and YARN configuration properties *yarn.scheduler.capacity.maximum-allocation-mb* and *yarn.scheduler.capacity.maximum-allocation-vcores*. Configuring Hadoop for launching containers necessitates the user's insight and expertise.

Inspired by work in [22], we implement an adaptiveConfig policy that interacts with YARN to obtain workload and cluster status. The configuration parameters are initiated at the onset of the cluster; YARN reads the job history server to obtain each job's status information, such as submission timestamps, resources required, etc. Next, it reads the *yarn-site.xml* file to obtain the status of the cluster resources such as maximum available *vcores* and *memory* on the node, Findex values, and the node label. Finally, it accesses the *fair-scheduler.xml* file to re-configure the scheduler's parameters. We modify these files to implement our adaptiveConfig policy. The Findex is used by the RM to dynamically set and assign the number of containers while considering the onboard processing power and memory availability on the node.

As an NM registers with the RM through the heartbeat message, the RM computes the number of available containers for each worker node based on the container-related properties defined in the configuration parameters. For an NM executing on a frugal node with an Findex larger than 1, it will assign only one container to execute on the node. Alternatively, for an Findex value of 2, up to a maximum of two containers would be assigned. For larger values of the Findex, the YARN default values set in, allowing more than two containers to be assigned to the NM.

The proposed adaptiveConfig scheduling policy enhances the Fair scheduling policy by facilitating adaptive resource allocation, dynamically adjusting to utilize the resources available on the physical nodes effectively. This approach ensures optimal resource utilization and enhances the overall efficiency of the system by intelligently allocating map tasks based on the varying capacities of individual nodes within the heterogeneous SBC-based cluster.

3.5. Task Locality and Prioritization

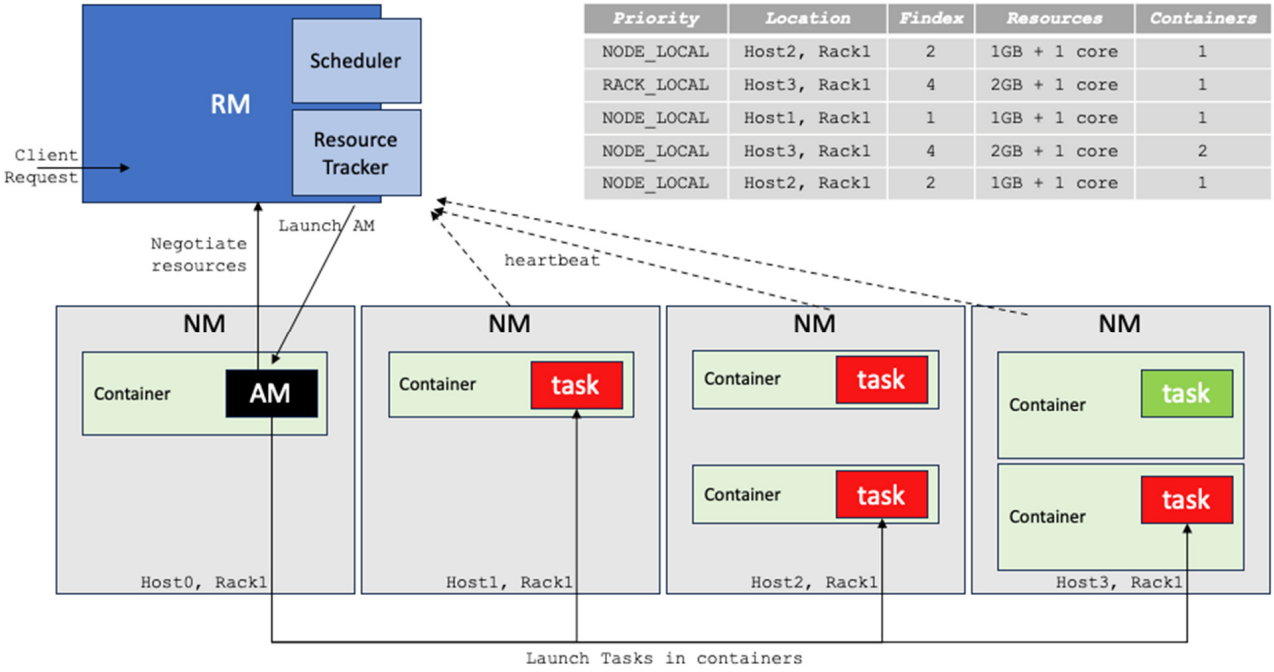
The scheduling policy aims to optimize resource utilization, minimize job completion time, and ensure fairness among users and applications sharing the cluster resources [23]. The inconsistent performance observed in Hadoop applications stems primarily from the performance gap among heterogeneous SBC nodes, which the native Hadoop framework fails to address adequately.

Unlike map tasks, there are no specific guidelines for assigning reduce tasks to cluster nodes [23–28]. Consequently, reduce tasks can be distributed across any node in the cluster, leading to significant performance discrepancies based on node capabilities. In essence, assigning reduce tasks to SBC nodes with limited computational power results in prolonged execution times for Hadoop MapReduce jobs, as map tasks on these nodes cannot fully leverage data locality.

Hadoop defines three priorities for data locality, namely NODE_LOCAL, RACK_LOCAL, and OFF_SWITCH [29]. NODE_LOCAL refers to the highest priority level for task scheduling. It means that the Hadoop scheduler tries to assign tasks to nodes where the data needed for computation is present, resulting in minimal data transfer across the network. RACK_LOCAL comes next in priority, where tasks are scheduled to nodes in the same rack as the required data, thus minimizing network traffic compared to off-rack assignments. Finally, OFF_SWITCH refers to the lowest priority level, where tasks are assigned to any available node regardless of its proximity to the data, resulting in potentially higher network overhead as data need to be transferred over longer distances. These priorities aim to optimize data locality and minimize network traffic for improved performance in Hadoop clusters [22].

In our proposal, the RM and AM are prioritized processes that need to execute on powerful SBCs with a higher priority. As these processes initiate at the onset of the cluster establishment, there is a higher probability that these processes would be assigned by the adaptiveConfig *policy* to powerful SBCs. However, the same cannot be said about application containers that are created to complete a MapReduce task [30]. As the number of tasks increases, there is no guarantee that the native Hadoop scheduler will assign fewer containers to a frugal node. It is quite possible that multiple map and reduce tasks would be assigned to a node hosting and possibly executing multiple containers on the same node, while other nodes in the cluster may have been assigned fewer containers or none at all. This uneven distribution of resources is quite common with native Hadoop.

Taking inspiration from the work presented in [31], the priority for any reduce task is set to RACK_LOCAL. This approach leverages the distributed nature of MapReduce processing, enabling efficient utilization of cluster resources and faster job completion times. This enhances placement or tasks in the clusters improving its overall parallelism. Figure 3 summarizes the proposed changes to the YARN information flow in a heterogeneous frugal SBC-based cluster.



Priority	Location	Findex	Resources	Containers
NODE_LOCAL	Host2, Rack1	2	1GB + 1 core	1
RACK_LOCAL	Host3, Rack1	4	2GB + 1 core	1
NODE_LOCAL	Host1, Rack1	1	1GB + 1 core	1
NODE_LOCAL	Host3, Rack1	4	2GB + 1 core	2
NODE_LOCAL	Host2, Rack1	2	1GB + 1 core	1

Figure 3. Information flow between various components of RM and NM.

4. Performance Evaluation and Results

This section presents the experimental evaluation and presents the empirical results.

4.1. Experimental Setup

We prepare our heterogenous frugal SBC cluster using 13 SBCs composed of 1 master node and 12 worker nodes. The master node is hosted on the best SBC at our disposal, i.e., the Raspberry Pi 5 assigned Findex = 4, *Node-label-Hi*. The worker nodes execute on 3 × Raspberry Pi5 Findex = 4, *Node-label-Hi*), 3 × Raspberry 3B Findex = 1, *Node-label-Low*), 3 × Odriod Xu4 (Findex = 2, *Node-label-Med*), and 3 × Rockpro64 (Findex = 2, *Node-label-Med*) SBCs. Details for these SBCs can be found in Table 1. Each SBC is fitted with a 64 GB SD card and is connected to a Gigabit Ethernet. A picture of the Hadoop cluster can be seen in Figure 4.

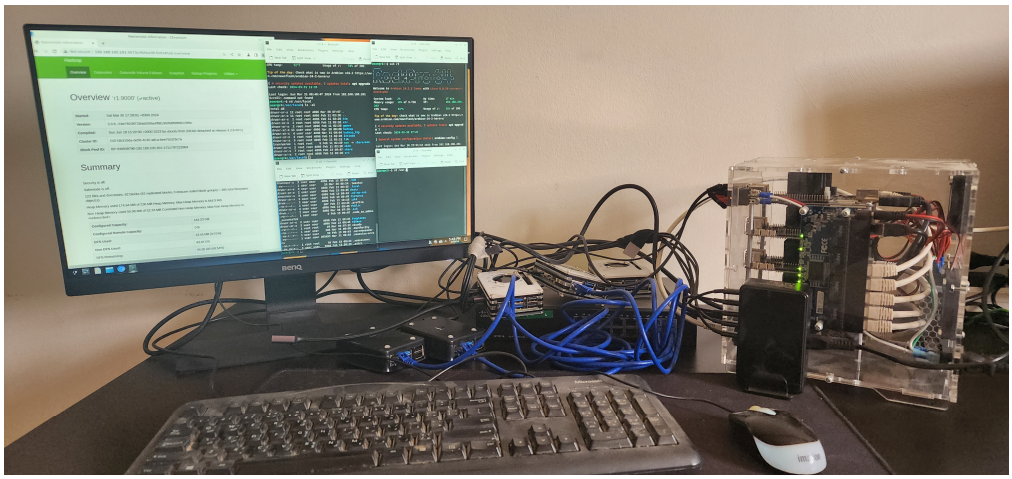


Figure 4. Heterogeneous Hadoop cluster built with frugal SBC devices, including RPi5, RPi3, Odroid Xu4, and RockPro64. One of the RPi5 serves as the master while the rest of the SBCs run the worker nodes. All the SBCs are connected to a Gigabit Ethernet switch.

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. A 4 GB swap space was reserved on all SBCs during installation. To initialize the Hadoop cluster, we used the *vcores* and *memory* limits provided in Table 2. The default Hadoop values for these properties always cause memory-related issues in the SBC clusters. The 4 GB swap space is useful for resource-frugal devices with limited onboard memory when running containers concurrently.

In this experimental study, we focus on task completion times for various map and reduce tasks. We also measure the CPU utilization, memory utilization, and network traffic. To ensure a comprehensive assessment, our evaluation will concentrate on workloads that are both CPU intensive and I/O intensive. Specifically, we will utilize two standard Hadoop micro-benchmarks, the WordCount and Terasort programs. WordCount is a CPU-intensive benchmark that involves counting the occurrences of words in a given dataset. It primarily stresses the computational capabilities of the system, making it suitable for evaluating CPU performance. On the other hand, Terasort is an I/O-intensive benchmark that focuses on sorting large volumes of data. This benchmark heavily exercises the input/output subsystem of the system, making it ideal for assessing I/O performance.

Through the evaluation of these benchmarks, we aim to evaluate the efficacy of the proposed changes to the YARN scheduling mechanism compared to the native YARN settings. This assessment involves analyzing how well the system manages tasks demanding substantial CPU processing and those reliant on intensive input/output operations. By focusing on these two distinct types of workloads, we can obtain a deeper understanding of the system's performance with regard to the placement of containers in the heterogenous SBC cluster.

Moreover, our evaluation extends to examining the influence of the Findex *value*/*Node-labels* on container placement within the cluster, taking into account the frugality levels of individual SBC nodes. Additionally, we delve into the consequences of the scheduling policy outlined in the preceding section, contrasting its effectiveness against the native Hadoop scheduling policies. This comprehensive experimental investigation aids in unraveling insights into the proposed system's adaptability to varying computational demands on frugal SBC-based clusters, thereby facilitating a more comprehensive assessment of its overall efficiency and efficacy.

4.2. Task Distribution in Native YARN vs. the Proposed Approach

The inherent behavior of the native Hadoop framework lacks discrimination in task assignment to worker nodes, disregarding their individual computational and memory capabilities. This indiscriminate allocation approach may inadvertently result in CPU-intensive tasks, such as RM and AM, being assigned to SBCs with lower performance capabilities within the cluster.

To comprehensively assess the ramifications of such task distribution, we conduct a detailed analysis focusing on the impact of heterogeneous node assignment on cluster performance. Leveraging the Terasort benchmark application, we closely monitor and evaluate how task distribution patterns influence overall cluster efficiency and resource utilization. Through this investigation, we aim to gain deeper insights into the dynamics of task allocation and its implications for workload management within heterogeneous SBC-based Hadoop clusters.

We establish four fundamental scenarios to delineate native Hadoop's task distribution, namely *best_cap*, *worst_cap*, *best_fifo*, and *worst_fifo*.

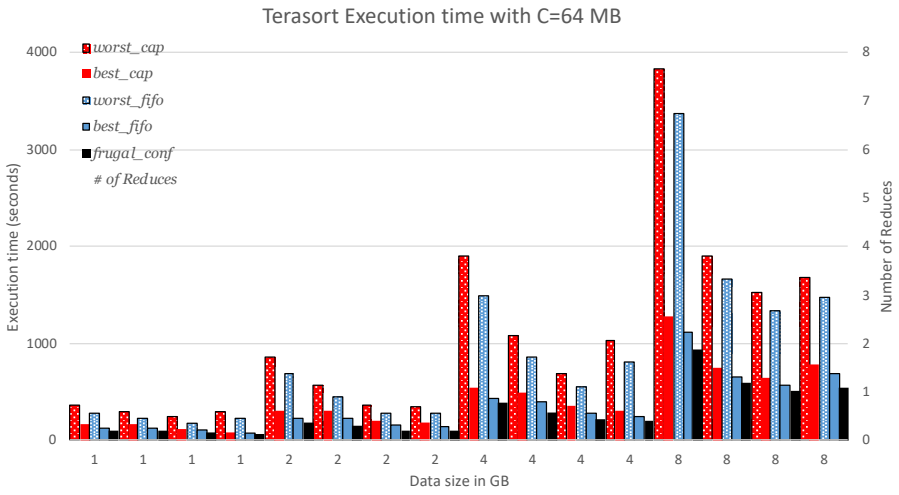
- Best Capacity Scenario (*best_cap*): In this setup, we allocate critical Hadoop components—the RM, AM, and reduce tasks—to high-performance SBCs. This assignment is managed using the default Capacity scheduler, ensuring that these critical tasks are handled by the most robust machines in the cluster.

- Worst Capacity Scenario (*worst_cap*): This scenario represents the opposite approach: RM, AM, and reduce tasks are allocated to lower-performance SBCs, the least capable work nodes. We continue using the Capacity scheduler, but this time with a focus on frugal SBCs. Like the *best_cap* scenario, the map tasks can be distributed to any available SBC, following the standard Capacity scheduling policy.
- Best FIFO Scenario (*best_fifo*): This setup is similar to the *best_cap* scenario in that RM, AM, and reduce tasks are assigned to high-performance SBCs. However, the key difference is that instead of using the Capacity scheduler, the FIFO (First-In-First-Out) scheduler is used for task allocation. This method assigns tasks in the order they are submitted.
- Worst FIFO Scenario (*worst_fifo*): Here, the RM, AM, and reduce tasks are directed to lower-performance SBCs, akin to the *worst_cap* scenario, but with a change in scheduling mechanism. The FIFO scheduler replaces the Capacity scheduler, organizing tasks based on their submission order, even if they are executed on less capable hardware. As with the *best_cap* and *worst_cap* scenarios, map tasks are distributed according to the default native YARN settings, which allows assignment to any available SBC.

These contrasting configurations in comparison to the proposed mechanism provide a clear framework for evaluating the impact of task assignment strategies on overall cluster performance.

Next, we run the Terasort benchmark application with various input data sizes and vary the number of reduce tasks to observe the time taken to complete the tasks. This allows us to compare the native Hadoop *best_cap*, *worst_cap*, *best_fifo*, *worst_fifo*, and the proposed *frugal_conf* and compare the execution runtimes.

Figure 5 shows the comparison of Terasort run times for *best_cap*, *worst_cap*, *best_fifo*, *worst_fifo*, and the *frugal_conf* settings. We show the comparison in terms of execution times for various settings running Terasort on the cluster with chunk sizes of 64 MB and 128 MB. The impact of the increasing number of reduce tasks can be observed in the Figure 5. For a single reduce task, the time taken for any size of dataset is the largest for *worst_cap*. It must be noted that as the number of reduce tasks increases, the execution time decreases proportionally; however, for *worst_cap*, the time increases due to the unavailability of powerful SBC nodes for reduce tasks. We observe a similar pattern with the *worst_fifo* policy.



(**a**)

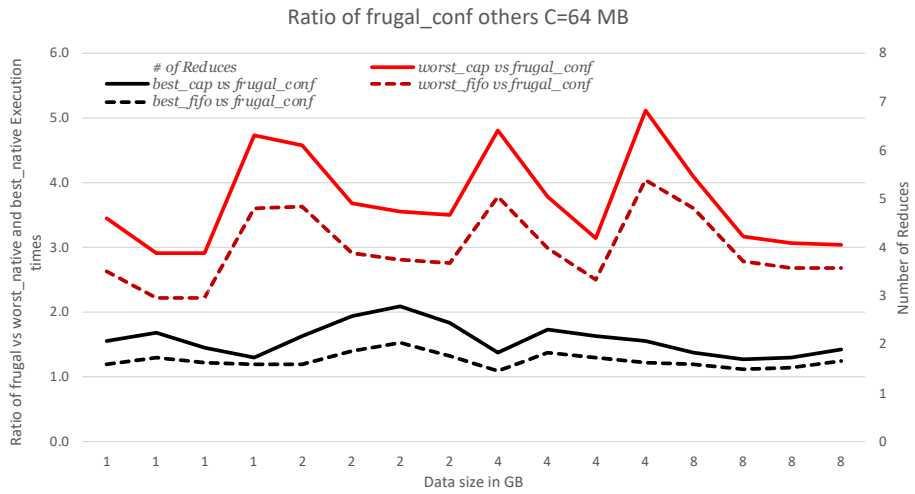
Figure 5. *Cont.*

(**b**)

Figure 5. Execution times of various Terasort jobs with varying dataset sizes, 1 GB, 2 GB, 4 GB, and 8 GB, and number of reduce tasks = 1, 2, 4, and 8. (**a**) Chunk size C = 64 MB. (**b**) Chunk size C = 128 MB.

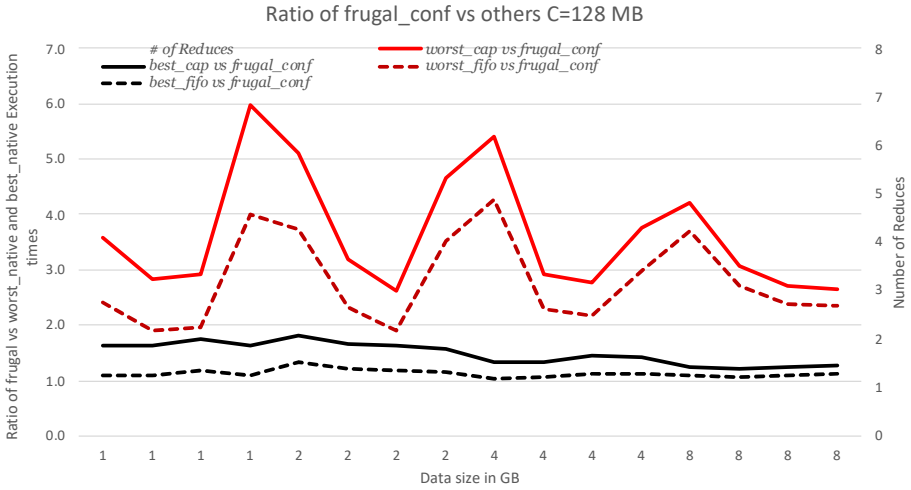
On the other hand, all the reduce tasks execute on powerful SBCs for the *best_cap* and *best_fifo* scenarios. As the number of reduce tasks exceeds the number of powerful available SBCs, i.e., eight reduce tasks, the execution time also increases. We note that this is because of native capacity scheduling multiple reduce tasks on the same node, causing delay in the overall execution time. In comparison, the proposed *frugal_conf* provides faster runtimes for all dataset sizes and numbers of reduce tasks. The proposed *frugal_conf* leverages the availability of the powerful SBCs to execute reduce tasks. Furthermore, as only one reduce task is allowed to execute on a powerful SBC, this results in a better uniform distribution of tasks across the cluster.

Figure 6a shows the comparison in terms of ratio of execution times, comparing *frugal_conf* with *best_cap*, *worst_cap*, *best_fifo*, and *worst_fifo*. The proposed *frugal_conf* executes on average, 270% and 62% faster than *worst_cap* and *best_cap*, respectively, for chunk size = 64 MB. *frugal_conf* executes, on average, 192% and 22% faster than *worst_fifo* and *best_fifo*, respectively, for chunk size = 64 MB.



(**a**)

Figure 6. *Cont*.



(**b**)

Figure 6. Comparison of Terasort execution time ratios of *frugal_conf* vs. *best_cap*, *worst_cap*, *best_fifo*, and *worst_fifo*. The data sizes are 1 GB, 2 GB, 4 GB, and 8 GB with number of reduce tasks = 1, 2, 4, and 8. (**a**) Chunk size C = 64 MB. (**b**) Chunk size C = 128 MB.

Figure 6b shows the comparison in terms of ratio of execution time comparing *frugal_conf* with *best_cap*, *worst_cap*, *best_fifo*, and *worst_fifo* for chunk size = 128 MB. The *frugal_conf* executes, on average, 267% and 53% faster than *worst_cap* and *best_cap*, respectively. *frugal_conf* executes, on average, 136% and 17% faster than *worst_fifo* and *best_fifo*, respectively.

4.3. Effect of adaptiveConfig Scheduling Policy on Task Distribution

To understand the effect of task distribution in the cluster using the Findex and node labels, we first analyze how the tasks are distributed on a single powerful SBC node. If the Findex value for an SBC node is 1, it implies that it is allowed to execute only one reduce task per node. Alternatively, an Findex value of 2 and 4 indicates that the system will assign up to two or four reduce tasks per SBC node, respectively. In the following experiment, we create three scenarios where (i) we assign Findex 1 to all nodes in the cluster; this will ensure that a max of one reduce task would execute on a node. (ii) We assign Findex 2 to all SBC nodes except for the weaker RPI 3B+ SBCs. This will allow a maximum of two reduce tasks to be co-located on a single SBC node. Finally, (iii) we assign Findex as presented in Table 3; this ensures that multiple reduce tasks are co-located on an SBC node.

Next, we execute Terasort and WordCount benchmarks on the cluster for various datasets of different sizes, 1 GB, 2 GB, 4 GB, and 8 GB. We also provide the number of reduce tasks to execute the Hadoop job. Figure 7 shows the execution runtimes of Terasort jobs for the various settings. For each data size, we see a decrease in execution time as the number of reducers increases from one to eight. This is the expected result of increased parallelism as more reducers allow for parallel processing of data, resulting in faster execution times. It is worth noting that the execution time for Scenario 3 is far less than Scenarios 1 and 2 for various chunk sizes and dataset sizes. This indicates that the proposed priority, along with Findex, ensures placement of the correct number of reduce tasks on each SBC node. As the data size increases, we generally observe an increase in execution time across all configurations. This is expected, as larger datasets require more processing time.

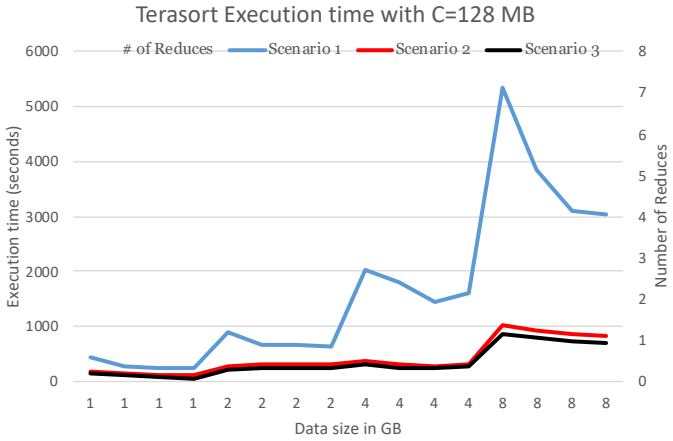


Figure 7. Comparison of Terasort execution times for Scenarios 1, 2, and 3. The data sizes are 1 GB, 2 GB, 4 GB, and 8 GB, with number of reduce tasks = 1, 2, 4, and 8. Chunk size C = 128 MB.

For the smaller 1 GB dataset with eight reduce tasks, the Scenario 3 configuration outperforms the Scenario 2 configuration by 96%. As the dataset size increases, with eight reduce tasks, the Scenario 3 configuration outperforms the Scenario 2 configuration by 27%. With 4 GB and 8 GB dataset sizes and eight reduce tasks, Scenario 3 is comparatively 18% and 13% better. On average, the Scenario 3 Terasort task executions show the lowest execution times, indicating that it is the most optimized configuration.

Figure 8 shows the execution runtimes of WordCount jobs for the various settings. As WordCount is a CPU-intensive application, it stress tests the CPU on the frugal SBC-based cluster. For larger datasets, e.g., with 8 GB Scenario 1 and Scenario 2, the configurations were not able to complete the task. Executing these tasks took in excess of 3 h of time, and hence these were terminated. For Scenario 1, with eight reduce jobs, it was not possible to complete the task as the policy restricts the cluster to execute multiple reduce tasks on each node. In some cases, the execution failed, which is attributed to the out-of-memory problem previously discussed. The Scenario 3 configuration was able to execute WordCount for all the experimental variations.

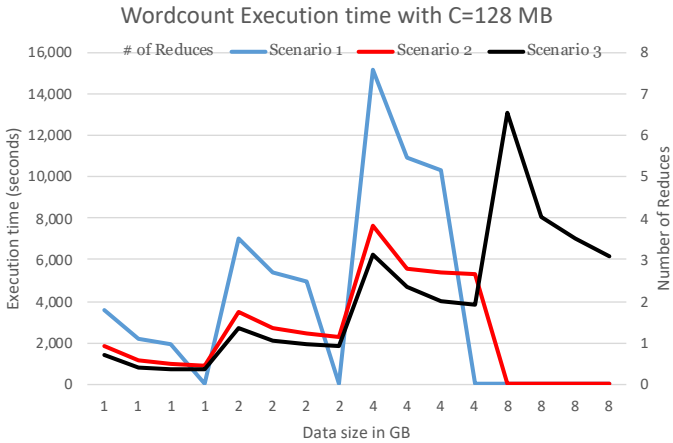


Figure 8. Comparison of WordCount execution times for Scenarios 1, 2 and 3. The data sizes are 1 GB, 2 GB, 4 GB, and 8 GB with number of reduce tasks = 1, 2, 4, and 8. Chunk size C = 128 MB.

For the 8 GB dataset with four reduce tasks, the Scenario 3 configuration outperforms the Scenario 2 configuration by 21%. For the 4 GB dataset with one, two, and four reduce tasks, Scenario 3 performs 32%, 27%, and 22% better, respectively, than Scenario 2. For the smaller dataset of 2 GB, with one, two, four, and eight reduce tasks, Scenario 3 performs 51%, 38%, 29%, and 22% better, respectively, than Scenario 2. Finally, for the smallest dataset with 1 GB, with one, two, four, and eight reduce tasks, Scenario 3 performs 59%, 42%, 31%, and 26% better, respectively, than Scenario 2.

5. Discussion and Future Directions

The concept of SBC-based clusters introduces a frugal approach to resource utilization in distributed computing environments. These clusters, often composed of devices like Raspberry Pi and Odroid Xu-4 possess limited processing power and memory compared to traditional server nodes. The frugality arises from the inherent constraints of these devices, which can impact their ability to efficiently execute concurrent MapReduce tasks. The study highlights that the performance of such clusters is notably affected by memory limitations, with devices like the Raspberry Pi 3B struggling due to their modest 1 GB RAM. This limitation necessitates adjustments in the Hadoop framework to accommodate the constraints of the SBCs, leading to the proposed changes in the YARN scheduling mechanism. The proposed changes aim to address the limitations posed by frugal SBC-based clusters. By introducing an Findex and adaptiveConfig policy, the redesign seeks to optimize resource allocation and enhance system efficiency. The Findex classifies SBC nodes based on their processing capacities and memory sizes, providing crucial information for container placement and task scheduling. Additionally, the adaptiveConfig policy dynamically adjusts resource allocation based on workload and cluster status, ensuring optimal utilization of available resources. These changes aim to mitigate performance bottlenecks caused by memory constraints and improve the overall efficiency of SBC-based clusters.

The suggested changes have significant implications for the performance and scalability of SBC-based clusters. By incorporating the Findex and adaptiveConfig policy into the scheduling mechanism, the clusters can adapt to the heterogeneous capacities of individual nodes more effectively. This adaptive approach enables better utilization of resources, mitigating the impact of frugality on cluster performance. Furthermore, the prioritization of tasks based on data locality and container status enhances parallelism and reduces job completion times. Overall, the proposed changes facilitate more efficient and resilient operation of SBC-based clusters, addressing the challenges posed by resource constraints. The results demonstrate the effectiveness of custom scheduling mechanisms in optimizing task distribution and improving overall cluster performance in a heterogeneous frugal SBC environment. By considering individual SBC capabilities through the Findex and adaptiveConfig policy, the study achieves better resource utilization and reduced task completion times.

With the emergence of powerful SBCs such as the Raspberry Pi 5, clusters comprised of these devices can significantly enhance both per-watt and per-dollar efficiency, thereby bolstering sustainability efforts. These SBCs are renowned for their energy efficiency, consuming minimal power while delivering respectable computational capabilities. In the experimental setup described, each SBC within the cluster is outfitted with a 64 GB SD card and connected via Gigabit Ethernet, ensuring minimal power consumption compared to conventional server configurations. The utilization of a frugal SBC cluster architecture optimizes resource utilization by employing only necessary components, thereby reducing overall energy consumption. Moreover, the adoption of frugal SBCs aligns with sustainability objectives by fostering more efficient resource utilization. By repurposing these low-power devices for cluster computing, organizations can prolong their lifespan, curbing electronic waste and promoting a more sustainable IT ecosystem. The cluster's focus on optimizing resource usage, exemplified by tailored configurations such as adjusting the number of reduce tasks per SBC node based on its Findex underscores the commitment to efficient resource allocation and sustainability.

The experimental setup's inclusion of a heterogeneous cluster comprising various SBC models allows for cost optimization by selecting models based on their price–performance ratio and specific workload demands. The cost of our cluster setup was USD 966 for the 13 devices, along with networking essentials (cables, Gigabit Switch) and SD card storage media. We noted that the cluster required approximately 76 W of power during WordCount execution. The overall power consumption ranged between 69 W and 78 W for the various experiments. To analyze the cluster performance in terms of performance-per watt and per dollar, we built a similar setup on a PC with an Intel i7-12700KF @ 12-Core processor

with 16 GB RAM and a 500 GB SSD. The power consumption for similar Terasort and WordCount jobs ranged between 112 W and 138 W. We also noted that the task execution times on PC were 31% and 56% faster for Terasort and WordCount jobs compared to the SBC-based cluster. The performance of the SBC-based cluster does not match that of a PC in terms of cost-effectiveness per dollar or per watt, mainly due to the fact that the previous generation RPi 3B nodes never reached the level of desktop PCs in either metric. Their overall performance remains notably lower compared to the latest generation RPi 5 nodes. At the moment, the cost of an RPi5 is approx. USD 80, and it is reasonable to anticipate that the prices for these devices will become lower in the near future. Heterogeneous SBC-based clusters comprised of the latest RPi 5 or upcoming generations of RPi nodes may present promising opportunities for enhancing big data processing performance metrics.

6. Conclusions

This experimental study underscores the efficacy of heterogeneous frugal SBC-based cluster for sustainable big data processing. The performance of resource-frugal nodes in the cluster is notably affected by memory limitations. These limitations necessitate adjustments in the Hadoop framework to accommodate the constraints. By introducing an Findex and adaptiveConfig policy, the redesign seeks to optimize resource allocation and enhance system efficiency. The Findex, together with Hadoop/YARN node-labels, serves as a crucial metric for categorizing SBC nodes based on their capabilities. By considering factors such as CPU speed and memory size, the Frugality Index leveraging the Hadoop node labels facilitates optimized resource allocation, ensuring that tasks are assigned to nodes best suited to handle them. The adaptiveConfig policy enhances the flexibility of the scheduler by dynamically adjusting resource allocation based on workload and cluster conditions. By optimizing resource allocation in real time, the policy enables SBC-based clusters to adapt to changing workloads and maintain high performance levels. Results show that by achieving faster execution times compared to traditional Hadoop configurations while consuming minimal power, the cluster maximizes computational output while minimizing energy expenditure. Further optimizations, such as the proposed scheduling mechanisms and parameter tuning, contribute to enhanced performance efficiency, enabling the cluster to achieve superior performance metrics relative to resource consumption. To evaluate the performance of the proposed work, we generated workloads of various sizes using two setting of chunk size 64 and 128 MB. Using the Hadoop benchmarks WordCount and Terasort, we compared the *frugal_conf* against four settings of Hadoop native FIFO and capacity schedulers, namely *best_cap*, *worst_cap*, *best_fifo*, and *worst_fifo*. The *frugal_conf* setting demonstrates significant performance improvements, executing 56% and 22% faster than the *best_cap* and *best_fifo* settings with a chunk size of 64 MB. It is also 53% and 17% faster than the *best_cap* and *best_fifo* settings with a chunk size of 128 MB.

The use of frugal SBCs aligns with sustainability goals by utilizing resources more efficiently. Instead of relying on high-power, energy-hungry servers, the cluster leverages multiple low-power SBCs, which collectively provide adequate computational capacity. By repurposing frugal SBCs for cluster computing, organizations can extend the lifespan of these devices, reducing electronic waste and contributing to a more sustainable IT ecosystem. The focus on optimizing resource usage, demonstrated by tailoring configurations such as the number of reduce tasks per SBC node based on its Findex ensures efficient utilization of computational resources, further enhancing sustainability.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

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

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

References

1. Castellanos-Rodríguez, Ó.; Expósito, R.R.; Enes, J.; Taboada, G.L.; Touriño, J. Serverless-like Platform for Container-Based YARN Clusters. *Future Gener. Comput. Syst.* **2024**, *155*, 256–271. [CrossRef]
2. Warade, M.; Schneider, J.-G.; Lee, K. Measuring the Energy and Performance of Scientific Workflows on Low-Power Clusters. *Electronics* **2022**, *11*, 1801. [CrossRef]
3. Johnston, S.J.; Basford, P.J.; Perkins, C.S.; Herry, H.; Tso, F.P.; Pezaros, D.; Mullins, R.D.; Yoneki, E.; Cox, S.J.; Singer, J. Commodity Single Board Computer Clusters and Their Applications. *Future Gener. Comput. Syst.* **2018**, *89*, 201–212. [CrossRef]
4. 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]
5. 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]
6. 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]
7. 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]
8. Jeyaraj, R.; Paul, A. Optimizing MapReduce Task Scheduling on Virtualized Heterogeneous Environments Using Ant Colony Optimization. *IEEE Access* **2022**, *10*, 55842–55855. [CrossRef]
9. 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]
10. Qureshi, B.; Koubaa, A. On Performance of Commodity Single Board Computer-Based Clusters: A Big Data Perspective. In *Smart Infrastructure and Applications: Foundations for Smarter Cities and Societies*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 349–375.
11. Apache Hadoop YARN. Available online: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html (accessed on 3 May 2024).
12. 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]
13. Thesma, V.; Rains, G.C.; Mohammadpour Velni, J. Development of a Low-Cost Distributed Computing Pipeline for High-Throughput Cotton Phenotyping. *Sensors* **2024**, *24*, 970. [CrossRef]
14. Veerachamy, R.; Ramar, R. Agricultural Irrigation Recommendation and Alert (AIRA) System Using Optimization and Machine Learning in Hadoop for Sustainable Agriculture. *Environ. Sci. Pollut. Res.* **2022**, *29*, 19955–19974. [CrossRef] [PubMed]
15. Setiyawan, A. Wireless Engine Diagnostic Tool Based on Internet of Things (IoT) With PiOBD-II Using Raspberry on Honda Jazz VTEC. *J. Phys. Conf. Ser.* **2022**, *2406*, 012028. [CrossRef]
16. Netinant, P.; Utsanok, T.; Rukhiran, M.; Klongdee, S. Development and Assessment of Internet of Things-Driven Smart Home Security and Automation with Voice Commands. *IoT* **2024**, *5*, 79–99. [CrossRef]
17. Chen, I.-T.; Tsai, J.-M.; Chen, Y.-T.; Lee, C.-H. Lightweight Mutual Authentication for Healthcare IoT. *Sustainability* **2022**, *14*, 13411. [CrossRef]
18. Basford, P.J.; Johnston, S.J.; Perkins, C.S.; Garnock-Jones, T.; Tso, F.P.; Pezaros, D.; Mullins, R.D.; Yoneki, E.; Singer, J.; Cox, S.J. Performance Analysis of Single Board Computer Clusters. *Future Gener. Comput. Syst.* **2020**, *102*, 278–291. [CrossRef]
19. Lim, S.; Park, D. Improving Hadoop Mapreduce Performance on Heterogeneous Single Board Computer Clusters. [CrossRef]
20. Nugroho, S.; Widiyanto, A. Designing Parallel Computing Using Raspberry Pi Clusters for IoT Servers on Apache Hadoop. *J. Phys. Conf. Ser.* **2020**, *1517*, 012070. [CrossRef]
21. 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]
22. Han, R.; Liu, C.H.; Zong, Z.; Chen, L.Y.; Liu, W.; Wang, S.; Zhan, J. Workload-Adaptive Configuration Tuning for Hierarchical Cloud Schedulers. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2879–2895. [CrossRef]
23. 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]
24. 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), Virtually, 21–24 November 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 26–30.
25. 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]
26. Fu, W.; Wang, L. Load Balancing Algorithms for Hadoop Cluster in Unbalanced Environment. *Comput. Intell. Neurosci.* **2022**, *2022*, 1–9. [CrossRef] [PubMed]
27. Singh, A.; Sandhu, R.; Mehta, S.; Giri, N.C.; Kuziakin, O.; Leliuk, S.; Saprykin, R.; Dobrozhan, A. A Comparative Study of Bigdata Tools: Hadoop Vs Spark Vs Storm. In Proceedings of the 2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, 2–6 October 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–5.

28. Banerjee, P.; Roy, S.; Sinha, A.; Hassan, M.d.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]
29. 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]
30. Ahmed, N.; Barczak, A.L.C.; Rashid, M.A.; Susnjak, T. A Parallelization Model for Performance Characterization of Spark Big Data Jobs on Hadoop Clusters. *J. Big Data* **2021**, *8*, 107. [CrossRef]
31. Tang, Z.; Zeng, A.; Zhang, X.; Yang, L.; Li, K. Dynamic Memory-Aware Scheduling in Spark Computing Environment. *J. Parallel Distrib. Comput.* **2020**, *141*, 10–22. [CrossRef]