Towards Improving YARN performance for Frugal Heterogeneous SBC-based Edge Clusters

Basit Qureshi *

Posted Date: 2 April 2024

doi: 10.20944/preprints202404.0154.v1

Keywords: Single Board Computers; Frugal edge computing; Hadoop; YARN; heterogeneous

Article

Towards Improving YARN Performance for Frugal Heterogeneous SBC-Based Edge Clusters

Basit Qureshi [1,2,*]

[1] Prince Sultan University, Riyadh 11586, Saudi Arabia
[2] University of Bradford, West Yorkshire, BD7 1DP, United Kingdom
* Correspondence: qureshi@psu.edu.sa;

Abstract: In the dynamic landscape of sustainable computing, use of edge devices is paramount for reducing the need for large-scale centralized data centers. By processing data locally, edge devices minimize the energy-intensive computing in data centers, improving the overall performance, cost-effectiveness whereas reducing the environmental impact. Edge devices may constitute edge clusters composed of resource frugal Single Board Computers (SBC) such as Raspberry Pi etc. The small form-factor and energy efficiency of these computers makes them ideal for processing large data on the edge. 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 Yet Another Resource Negotiator (YARN) scheduling mechanism to address these challenges. Our proposed changes include the introduction of a Frugality Index and an adaptiveConfig policy. The Frugality Index categorizes SBC nodes based on their capabilities, enabling intelligent resource allocation. The adaptiveConfig policy dynamically adjusts resource allocation in response to workload and cluster conditions, enhancing system efficiency. Additionally, we introduce a fetch_threshold for reduce tasks to improve task prioritization based on locality and data processing efficiency. We evaluate our approach using a 13-node SBC cluster and conduct experiments with CPU-intensive and IO-intensive Hadoop benchmarks. The results demonstrate significant performance improvements compared to native YARN settings, with execution times 4.7 times faster than the worst_native and 1.9 times faster than the best_native scenarios. Furthermore, the proposed adaptiveConfig policy implementing the frugality index and a fetch_threshold outperforms the native YARN by 5.86 times and 1.79 times in Terasort and wordcount executions respectively. Our findings underscore the effectiveness of our approach in managing the heterogeneous nature of SBC clusters and optimizing performance across various hardware configurations. The adaptive policies prove well-suited to the frugal SBC-cluster context, yielding enhanced outcomes and paving the way for sustainable big data processing initiatives.

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

1. Introduction

In the realm of sustainability and environmental conservation, the utilization of low-cost single board computers (SBCs) in edge devices stands out as a beacon of innovation and efficiency. These compact computing devices offer a myriad of benefits that extend beyond conventional computing paradigms. 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 [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

 doi:10.20944/preprints202404.0154.v1

minimizing energy consumption, SBC-based edge devices contribute to overall energy conservation efforts, thereby reducing carbon emissions and lessening the strain on the environment [2]. Edge clusters, comprised of diverse SBCs with varying processing capabilities, memory, and power requirements, present a unique opportunity to leverage distributed computing resources in environmentally conscious ways.

The compact form factor of SBCs enables the development of small-scale, localized computing solutions tailored to specific environmental challenges. These edge devices can be strategically deployed in various settings, ranging from agricultural fields to urban infrastructure, to collect and analyze data in real-time. By processing data at the edge, without the need for continuous connectivity to centralized servers, SBC-based devices minimize latency and bandwidth requirements while enhancing overall system responsiveness. V. Thesma et. al., in [3] 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 [4] present 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 [5] developed a 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 repairs 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 [7] 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. Authors in [8] analyze the impact of Lightweight Mutual Authentication for Healthcare IoT. The proposed technique significantly improves the disadvantages of IoT devices that lack computing power.

Over the past decade, Apache Hadoop has become a leading framework for big data processing [9]. Hadoop, a robust framework designed for distributed storage and processing of vast datasets, serves as a cornerstone in fostering sustainability initiatives across diverse domains. Its distributed computing model enhances energy efficiency by enabling parallel processing of data across multiple nodes within a cluster. Lately, researchers in [9–14] 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 context including agriculture, smart cities, smart homes, healthcare etc. Qureshi et. al. in [11] developed a heterogenous cluster of 20 SBCs including Raspberry Pis and Ordoid Xu-4 for data analytics using Hadoop. They conduct 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 cluster was inferior to Ordoid Xu-4 machines due to the frugal nature of the devices. Lee in [12] present 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 5 latest model SBC can successfully process workload of a few tera-bytes. Neto et.al in [13] analyze the performance of Raspberry Pi based cluster using various benchmark 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 [14] also design a parallel computing framework using raspberry Pi clusters for IoT services and applications. The proposed framework uses Hadoop HDFS for data storage and processing.

Based on the preceding studies, it is evident that employing SBC-based clusters for big data processing with Hadoop offers viable and sustainable solutions for diverse applications. The presence of heterogeneous SBC clusters within the Hadoop framework introduces fresh challenges stemming from disparities in computing resources across individual nodes. Native Hadoop fails to adequately address the diversity among cluster nodes, leading to notable discrepancies in

 doi:10.20944/preprints202404.0154.v1

performance or, more critically, recurrent node failures within heterogeneous SBC-based Hadoop clusters. At its core, Yet Another Resource Negotiator (YARN) serves as a resource management and job scheduling framework in Apache Hadoop, facilitating the efficient allocation of computational resources across a cluster. However, traditional approaches to YARN optimization may not fully account for the characteristics of frugal heterogeneous edge clusters, where SBCs operate under constraints of computational power, memory, and network connectivity. Thus, there is a pressing need to explore novel strategies and techniques tailored to the specific challenges and opportunities presented by SBC-based edge computing environments.

In this study, we propose modifications to the YARN scheduling mechanism aimed at enhancing system efficiency in SBC-based clusters. These changes involve the introduction of a Frugality-Index and an *adaptiveConfig* policy. The Frugality-Index serves as a pivotal metric for categorizing SBC nodes according to their capabilities, incorporating factors such as CPU speed and memory size. This index facilitates intelligent resource allocation, ensuring tasks are assigned to nodes best equipped to handle them. Additionally, the *adaptiveConfig* policy enhances the YARN scheduler's flexibility by dynamically adjusting resource allocation in response to workload and cluster conditions. This real-time optimization enables SBC-based clusters to adapt to evolving workloads while maintaining high performance levels. Furthermore, the introduction of a *fetch_threshold* for reduce tasks enhances task prioritization and overall data processing efficiency.

We construct a SBC cluster composed of 13 SBC devices and conduct various experiments to test the proposed scheduling mechanism using CPU-intensive and IO-intensive Hadoop benchmarks against native YARN settings. The proposed settings demonstrate significant performance improvements, executing 4.7 times faster than the *worst_native* and 1.9 times faster than the *best_native* scenarios. In terms of Terasort execution, Scenario3 outperforms Scenario1 by 5.86 times and Scenario2 by 1.79 times. Additionally, setting the *fetch_threshold* to 0.05 achieves 1.23 times faster runtimes for configurations leveraging higher level of parallelism. Our findings indicate the efficacy of our approach in managing the heterogeneous cluster nature and performing well across standard CPU-Intensive and IO-Intensive Hadoop benchmark applications. Additionally, we ascertain that the adaptive policies are well-matched to the frugal SBC-cluster context, yielding enhanced outcomes across various hardware configurations, including newer high-performance models and older, slower SBCs.

The rest of the paper is organized as follows. Section 2 presents relevant work and background. Section 3 details the re-designed architecture of the YARN based on the proposed policy framework. 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

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.

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 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,

 doi:10.20944/preprints202404.0154.v1

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, are cost-effective, energy efficient and are widely accessible and have been used in various studies. A major drawback with earlier generation RPi was the computational capacity as highlighted in our earlier work in [11]. With newer models 3B+, 4B and 5th 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 display. We also use Odriod XU-4 [1]SBCs that use Samsung Exynos Octa core ARM processor with a 2GHz 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 [2]is another SBC used in this work. It's 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 4GB LPDDR4 system memory and 128Mb SPI boot Flash. All of these SBCs support microSD Cards for storage with varying sizes including 64GB. Odroid Xu4 and Rockpro64 also support the faster eMMC modules.

Table 1. Specifications of various SBC 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	1.8GHz Hexa Rockchip RK3399 ARM Cortex A72 and 1.4 GHz Quad Cortex-A53	1.4GHz 64-bit quad-core ARM Cortex-A53	Exynos5 Octa ARM Cortex-A15 Quad 2Ghz and Cortex-A7 Quad 1.3GHz
Memory	8GB LPDDR4X-SDRAM	4GB LPDDR4-SDRAM	1GB LPDDR3-SDRAM	2GB DDR3
Ethernet	Gigabit Ethernet	Gigabit Ethernet	300Mbit/s	Gigabit Ethernet
GPU	VideoCore VII 800MHz	Mali-T860 GPU 700MHz	VideoCore IV 400MHz	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

[1] Odroid Xu-4 https://www.odroid.co.uk/hardkernel-odroid-xu4/odroid-xu4

[2] Pine 64 RockPro64 https://pine64.com/product/rockpro64-4gb-single-board-computer/

 doi:10.20944/preprints202404.0154.v1

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 report to the RM through periodic *heatbeat* messages. Tasks affected by node failures are rescheduled on available nodes, and 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.

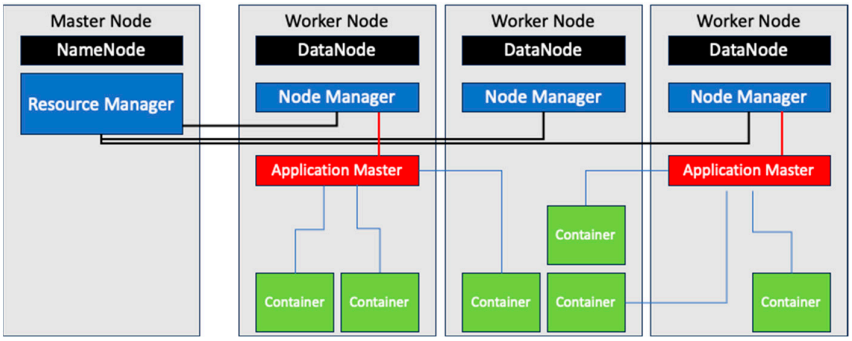


Figure 1. Hadoop YARN architecture and service flow.

In this work we propose changes to the YARN architecture so that it is able to discern frugal SBC based nodes in the cluster. The proposed changes are designed to improve the resource scheduling policies focusing on optimal placement of resources in the cluster. In order to evaluate the proposed approach, we implement a SBC based cluster consisting of four types of SBC devices. Our results demonstrate that the proposed approach effectively adapts to the heterogeneous nature of the cluster and perform well for standard *CPU-Intensive* and *IO-Intensive* Hadoop benchmark applications. Furthermore, our findings illustrate that the adaptive policies are well-suited to the frugal SBC-cluster environment, yielding improved outcomes across both higher-grade hardware and older, slower SBC models.

3. Proposed Scheduling Mechanism for Frugal SBC-Based Clusters

3.1. Motivation and Limitations

In the native Hadoop framework, there exists no inherent mechanism for determining the specific capacities of individual nodes, such as CPU processing capabilities, or physical memory availability, etc. 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.

 doi:10.20944/preprints202404.0154.v1

In [11], the authors observed that the performance of 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 task. 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 bottleneck due to memory restrictions. It was able to handle up to two containers per node/device.

However, the native YARN settings do discern the limited capabilities of these devices. When the number of containers exceeds two on a 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 irresponsive 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 only one container to execute on the frugal RPi devices in the cluster ensuring that the application does not crash. A similar observation is also made by the authors in [12] where the authors run in to 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.

In this section, we redesign the YARN scheduling mechanism to align with the frugal-SBC resources in the cluster. We define a frugality-index that classifies frugal SBC nodes based on their onboard processing capacities and memory size. Using Hadoop Remote Procedure Calls (RPC), the *frugalityIndex* is passed as 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 *frugalityIndex* and proposed a *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 Map and Reduce tasks according to the heterogeneous capacities of nodes within the SBC-based cluster. The following details these proposed changes to the YARN design.

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

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

 doi:10.20944/preprints202404.0154.v1

3.2. Frugality Index

The native Hadoop framework lacks any mechanism to discern the container placement on nodes based on their specific physical computational capacities or physical memory space. As mentioned earlier, it is evident that the physical memory capacities of nodes play a crucial role in influencing the concurrent execution of MapReduce tasks.

The NM on each node in the cluster determines the frugality index (Findex) based on the local device/ Node's physical characteristics. Table 3 presents a Findex guideline for various SBC used in this study. 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 scehduling policies to place containers on the various worker-nodes.

Table 3. Frugality Index guideline.

FIndex	Device	CPU	Memory
4	Raspberry Pi 5	2.4 GHz	8 GB
3	Raspberry Pi 4	1.5 GHz	4 GB
3	Pine64 Rockpro64	1.8 GHz	4 GB
2	Odroid Xu4	2.0 GHz	2 GB
1	Raspberry Pi 3B	1.4 GHz	1 GB
1	Raspberry Pi 2	900 MHz	1 GB

3.3. 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 request to the scheduler. Based on the provided parameters, the Scheduler requests *ResourceTracker* to launch the AM. It finds 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. NM periodically update the RM to inform about their available resources through a process called the *heartbeat* mechanism.

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 RM receives these *heartbeat* messages from all active NM in the cluster. Based on its resource allocation decision, the RM communicates with specific NM to allocate containers for executing job tasks. Each container is launched with the specified resource allocation, and tasks within the containers begin execution. Throughout the job's execution, NM 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 NM. 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.

 doi:10.20944/preprints202404.0154.v1

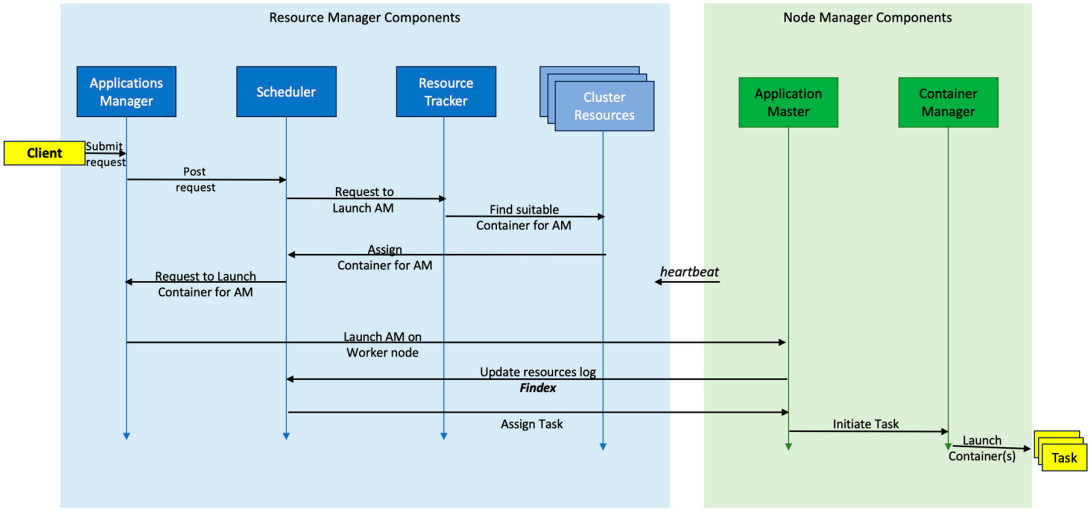


Figure 2. Information flow between various components of RM and NM in modified YARN.

3.4. Adaptive Fair Scheduling Scheme

Native YARN offers three distinct scheduling policies: FIFO, Capacity, and Fair [15]. 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 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 [16,17], 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 jobs 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. Finally, it accesses the capacity-scheduler.xml or fair-scheduler.xml file to re-configure the schedulers 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 a NM registers with the RM, through the heartbeat message, RM computes the number of available containers for each worker nodes based on the container related properties defined in the configuration parameters. For a NM executing on a frugal node with *Findex* larger than 1, it will assign only one container to execute on the node. Alternatively, for a *Findex* value 2, up to a maximum of two containers would be assigned. For larger values of *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 heterogenous SBC based cluster.

 doi:10.20944/preprints202404.0154.v1

3.5. Tasks 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 [18]. 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 AM and reduce tasks to cluster nodes. Consequently, AM and 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 [19]. NODE_LOCAL refers to the highest priority level for task scheduling. It means that 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 needs to be transferred over longer distances. These priorities aim to optimize data locality and minimize network traffic for improved performance in Hadoop clusters [20].

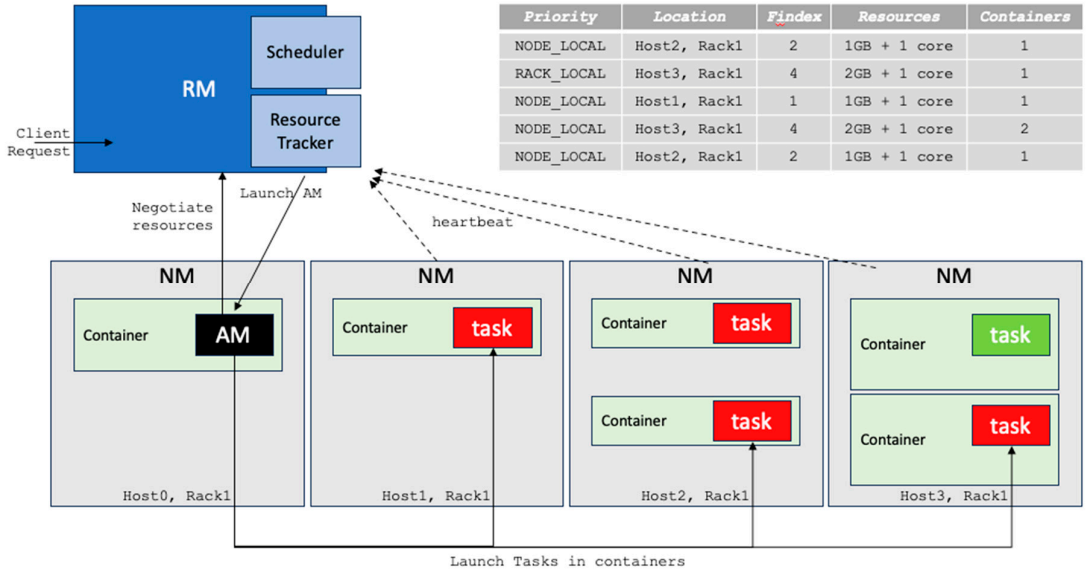
In our proposed YARN re-design, the RM and AM are processes that need to execute on powerful SBC with a higher priority. We define the NODE_LOCAL(HIGH) priority that would be assigned to these processes on any available powerful SBCs. As these processes initiate at the onset of the cluster establishment, there is a higher probability that these processes would be assigned to powerful SBC. However, the same cannot be said about application containers that are created to complete a MapReduce Task [21]. As the number of tasks increase, there is no guarantee that the native Hadoop scheduler would 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.

To alleviate this concern and to improve uniform distribution of tasks across the cluster, we look at the state of the container. The status of any container in Hadoop can be any of ALLOCATED, ACTIVE, PENDING, COMPLETED or KILLED. A container is considered ACTIVE when it has been allocated resources and is currently executing tasks assigned to it. During this state, the container is actively processing data or running computations as part of a job. A container is in a PENDING state when it has been requested by an application but has not yet been allocated resources to start execution.

Containers typically enter the pending state while the RM processes resource requests and determines where to allocate resources within the cluster. During the shuffle and sort phase, intermediate key-value pairs are streamed from map task outputs to the disks of the nodes where the reduce tasks will be executed. As these intermediate key-value pairs arrive at the reduce node, they are immediately available for processing by the reduce task. The reduce task can start processing the intermediate data as soon as a predefined threshold of data is available, typically referred to as the *fetch_threshold*. This threshold ensures that the reduce task has enough data to begin its processing efficiently without waiting for the entire dataset to be transferred. Once the *fetch_threshold* is reached, the reduce task initiates its processing logic, which involves grouping, sorting, and reducing the intermediate data to produce the final output. Furthermore, the priority for any reduce task is set to RACK_LOCAL; i.e. no two reduce tasks would execute on the same physical node. By starting to work on available intermediate data early, reduce tasks can overlap their computation with the ongoing data transfer, thus reducing idle time and improving overall job throughput. 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

 doi:10.20944/preprints202404.0154.v1

its overall parallelism. Figure 3 summarizes the proposed changes to the YARN information flow in a heterogenous 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 in modified YARN.

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 SBC composed of one master node and 12 worker nodes. The master node would be hosted on the best SBC at our disposal, i.e. Raspberry Pi 5 assigned *Findex*=4. The worker nodes would execute on 3x Raspberry Pi5 (*Findex*=4), 3x Raspberry 3B (*Findex*=1), 3x Odriod Xu4 (*Findex*=2) and 3x Rockpro64 (*Findex*=3) SBC. 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 schematic diagram can be seen in figure 4. The Ubuntu 22.04.4 LTS 64-bit Operating System (OS) for ARM processors was installed on each SDCard. A 4 GB swap space was reserved on all SBC during installation. We opted not to install a Graphic User Interface (GUI) like the GNOME desktop on Ubuntu, this was to maximize the available resources for YARN. Hadoop version 3.3.6 was installed on each node. 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 4GB swap space would be useful for resource-frugal devices with limited onboard memory when running containers concurrently.

 doi:10.20944/preprints202404.0154.v1

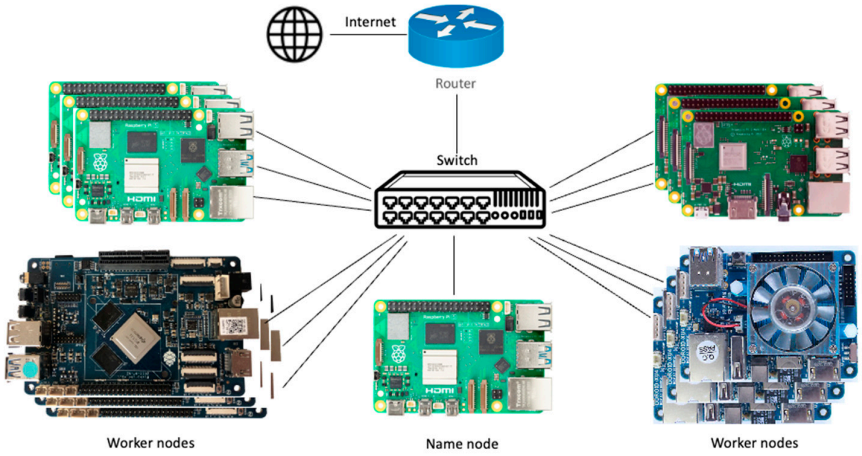


Figure 4. Heterogeneous Hadoop cluster built with frugal SBC devices including RPi5, RPi3, Odroid Xu4 and RockPro64.

In this experimental study we would be focusing on task completion times for various map and reduce tasks. We would also be measuring the CPU utilization, memory utilization, 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 benchmarks: 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 regards to the placement of containers in the heterogenous SBC cluster.

Moreover, our evaluation extends to examining the influence of the Frugality Index *Findex value* 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. Furthermore, we scrutinize the implications of prioritizing container placement for RM, NM, AM, and Reduce tasks. This comprehensive experimental investigation aids in unraveling insights into the proposed system's adaptability to varying computational demands on frugal SBC-based cluster, 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

 doi:10.20944/preprints202404.0154.v1

gain deeper insights into the dynamics of task allocation and its implications for workload management within heterogeneous SBC-based Hadoop clusters.

We establish two fundamental scenarios to delineate native Hadoop's task distribution: Firstly, in the *best_native* scenario, RM, AM, and Reduce tasks are allocated to robust SBCs only. Conversely, in the *worst_native* scenario, RM, AM, and Reduce tasks are dispatched to frugal SBC work nodes only. The map tasks are assigned to any available SBC device as in default native YARN scheduler settings. 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_native*, *worst_native* and the proposed YARN framework designed for frugal SBC-based clusters, *frugal_conf* runtimes.

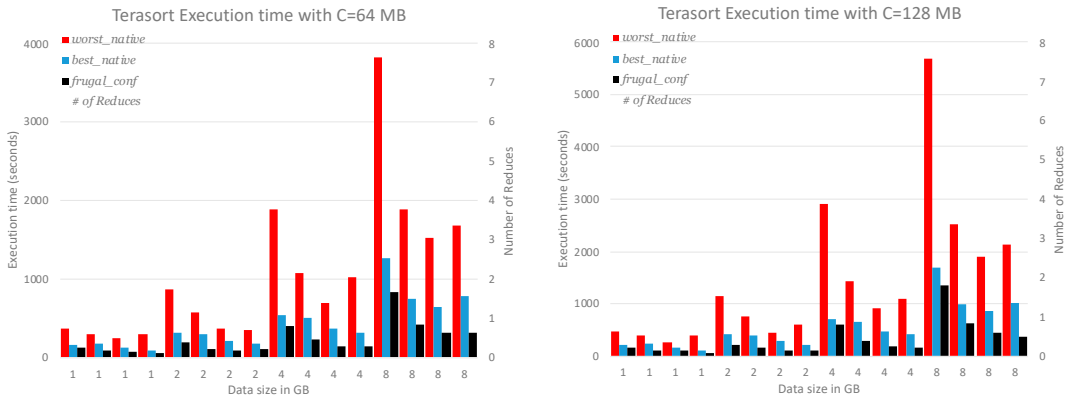


Figure 5. Execution times of various Terasort jobs with chunk size C= 64MB and 128MB with varying dataset size 1GB, 2GB, 4GB and 8GB and Number of Reduce tasks = 1, 2, 4 and 8. .

Figure 5 shows the comparison of Terasort run times for *best_native*, *worst_native,* and *frugal_conf* settings. We show the comparison in terms of execution times for various settings running Terasort on the cluster with chunk sizes 64 MB and 128 MB. The impact of the increasing number of reduce tasks can be observed in the figure. For a single reduce task, the time taken for any size of dataset is the largest for *worst_native*. It must be noted that as the number of reduce tasks increase, the execution time decreases proportionally, however for *worst_native*, the time increases due to unavailability of powerful SBC nodes for reduce tasks. On the other hand, all the reduce tasks execute on powerful SBCs for *best_native* scenario. As the number of reduce tasks exceed the number of powerful available SBC, i.e. for 8 reduce tasks, the execution time also increases. We note that this is because of native RM scheduling multiple reduce tasks on the same node causing delay in overall execution time. In comparison the proposed *frugal_conf* provides faster runtimes for all dataset sizes and number of reduce tasks. The proposed *frugal_conf* leverages the availability of 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 6 shows the comparison in terms of ratio of execution time comparing *frugal_conf* with *worst_native* and *best_native*. The proposed *frugal_conf* executes on average of 4.6x and 2.0x faster than *worst_native* and *best_native* respectively for chunk sizes = 64MB. For larger chunk size = 128, *frugal_conf* executes 4.7x and 1.9x faster than the *worst_native* and *best_native* scenarios. These results show that the proposed *frugal_conf* outperforms the native YARN baseline best and worst settings.

 doi:10.20944/preprints202404.0154.v1

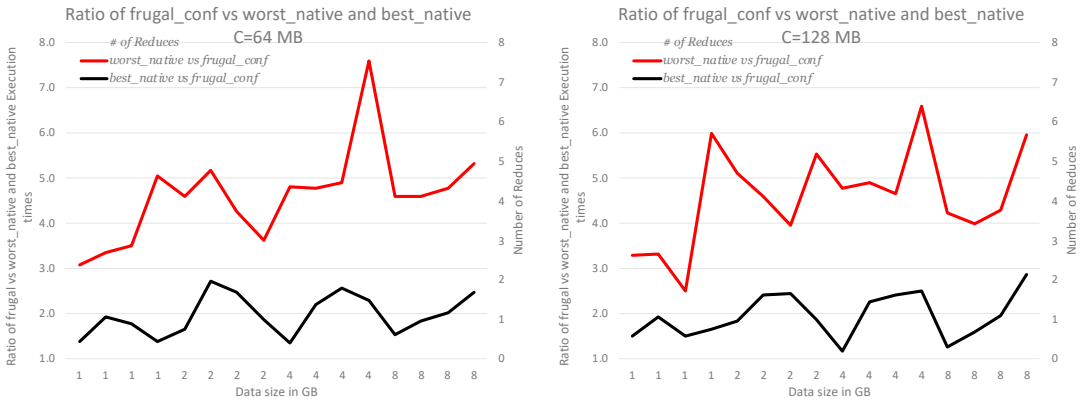


Figure 6. Comparison of Terasort Execution times ratios of *frugal_conf* vs *worst_native* and *frugal_conf* vs *best_native* for chunk sizes C=64MB and C=128MB. The datasizes are 1GB, 2GB, 4GB and 8GB with Number of Reduce = 1, 2, 4 and 8.

4.3. Effect of adaptiveConfig Scheduling Policy on Task Distribution

To understand the effect of *Findex* and the proposed NODE_LOCAL(HIGH) priority in the cluster, we first analyze how the tasks are distributed on a single powerful SBC node. If the *Findex* value for a SBC node is 1, it implies that it is allowed to execute only one AM or reduce task per node. Alternatively, a *Findex* value of 2 and 4 indicates that the system will assign up to 2 or 4 reduce tasks per SBC node respectively. This setting may allow AM to co-locate with reduce tasks on powerful SBCs.

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 AM or reduce task would execute on a node. *ii)* We assign *Findex* 2, to all SBC nodes besides RPI 3B+ SBCs. This will allow a maximum of two AM or reduce tasks to be co-located on a single SBC node. Finally, *iii)* we assign *Findex* as presented in Table 3, this ensures that AM and multiple reduce tasks are co-located on a SBC node. Next, we execute Terasort and wordcount benchmark on the cluster for various datasets of different sizes 1GB, 2GB, 4 GB and 8GB. We also provide the number of reduce tasks to execute the Hadoop job.

Table 4 shows the execution runtimes of Terasort jobs for the various settings. For each data size and chunk size, we see a decrease in execution time as the number of reducers increases from 1 to 8. This is 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 NODE_LOCAL(HIGH) priority along with Findex ensures placement of correct number of AM and 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. On average, the Scenario3 Terasort task executions show the lowest execution times, indicating that it is the most optimized configuration. For 8GB dataset with 8 reduce tasks, Scenario3 configuration outperforms scenario1 configuration by 5.86x, whereas it outperforms Scenario2 configuration by 1.79x. Similarly, Scenario 2 also outperforms the Scenario1 by 3.65x.

Table 4. Terasort execution times for *Findex* scenarios with various Datasize and Reduce jobs.

		Terasort execution time (seconds)					
		Chunk size 64			**Chunk size 128**		
# of Reduce	Data size (GB)	Scenario1	Scenario2	Scenario3	Scenario1	Scenario2	Scenario3
1	1	392.3	163.1	132.1	451.1	171.3	129.5
2	1	235.3	144.7	117.2	270.6	151.9	114.9
4	1	219.6	114.3	92.6	252.5	120.0	66.9

 doi:10.20944/preprints202404.0154.v1

8	1	201.4	109.8	88.9	231.6	115.3	58.7
1	2	861.0	273.2	221.3	887.1	275.9	216.9
2	2	693.1	289.6	234.6	679.5	292.5	229.9
4	2	615.4	293.8	238.0	668.6	293.1	233.2
8	2	598.4	291.6	236.2	645.7	294.5	231.5
1	4	1989.1	351.4	305.7	2015.6	365.5	299.6
2	4	1673.3	298.3	259.5	1798.4	310.2	254.3
4	4	1498.1	274.5	238.8	1456.1	269.3	234.0
8	4	1613.7	319.6	278.1	1598.7	323.1	272.5
1	8	5193.9	1025.6	892.3	5341.9	1016.3	874.4
2	8	3819.2	916.5	797.4	3857.4	934.8	781.4
4	8	3189.1	856.1	744.8	3093.4	873.2	729.9
8	8	3091.8	813.5	707.7	3030.0	829.8	693.6

Table 5 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 8GB Scenario1 and Scenario2 configurations were not able to complete the task. Executing these tasks took excess of 3 hours of time, hence these were terminated. For Scenario1, with 8 reduce jobs, it was not possible to complete the task as the policy restricts the cluster to execute multiple AM and reduce tasks on each node. In some cases, the execution failed which is attributed to the out of memory problem previously discussed. Scenario3 configuration was able to execute wordcount for all the experiment variations. For 4GB dataset with 4 reduce tasks, Scenario3 configuration outperforms scenario1 configuration by 2.63x, whereas it outperforms Scenario2 configuration by 1.29x. Scenario2 configuration also outperforms the Scenario1 by 1.98x.

Table 5. Wordcount execution times for *Findex* scenarios with various Datasize and Reduce jobs. *denotes the job was not completed in the max allowed time.

		WordCount execution time (seconds)					
		Chunk size 64			**Chunk size 128**		
# of Reduces	Data size (GB)	Scenario1	Scenario2	Scenario3	Scenario1	Scenario2	Scenario3
1	1	3089.1	1729.9	1401.2	3552.5	1816.4	1373.2
2	1	1891.6	1059.3	858.0	2175.3	1112.3	840.9
4	1	1651.4	924.8	749.1	1899.1	971.0	734.1
8	1	*	875.1	708.8	*	918.9	674.1
1	2	6103.7	3418.1	2768.6	7019.3	3452.3	2713.3
2	2	4714.3	2640.0	2138.4	5421.4	2666.4	2095.6
4	2	4309.1	2413.1	1954.6	4955.5	2437.2	1915.5
8	2	*	2289.7	1854.7	*	2312.6	1817.6
1	4	13173.8	7377.3	6418.3	15149.9	7672.4	6289.9
2	4	9513.4	5327.5	4634.9	10940.4	5540.6	4681.3
4	4	8963.1	5219.4	3953.0	10307.6	5428.2	3992.5
8	4	*	5069.1	3761.0	*	5271.9	3798.6
1	8	*	*	12915.3	*	*	13044.5
2	8	*	*	8194.0	*	*	8030.1
4	8	*	*	7149.0	*	*	7006.0
8	8	*	*	6328.0	*	*	6201.4

4.4. Effect of fetch_threshold Values

In section 3.5, we defined a *fetch_threshold* that initiates a reduce task to start processing of the intermediate data as soon as the data is available. This threshold ensures that the reduce task has

 doi:10.20944/preprints202404.0154.v1

enough data to begin its processing efficiently without waiting for the entire dataset to be transferred. In this analysis, we investigate the influence of the *fetch_threshold* parameter on the parallel execution of Terasort and Wordcount applications. Our aim is to discern how this parameter affects the level of parallelism within the proposed scheduling mechanism.

The results in figure 7 show varying trends in Terasort execution time with different values of *fetch_threshold*. We execute the terasort benchmark with 4 and 8 reduce to understand the impact of *fetch_threshold* on the parallelism on the cluster. For Scenario3, with a *fetch_threshold* of 0.05, there is a consistent decrease in execution time as the number of reducers increases. This shows that the reduce tasks would be initiated as soon as 5% of intermediate data is available therefore improving the parallelism resulting in reduced execution time. However, as *fetch_threshold* increases, this trend is not consistently observed. For larger data sizes and higher numbers of reducers, the execution time tends to decrease initially and then stabilize or increase slightly. It has a negligible effect with scenarios with a smaller number of reduce tasks.

Across all dataset sizes, reducing the *fetch_threshold* from 0.3 to 0.05 consistently reduces execution times. This is because reducing the threshold allows reduce tasks to start processing intermediate data earlier, improving parallelism and reducing overall execution times. Similar to the case of a single reduce task, decreasing the *fetch_threshold* generally results in shorter execution times for Terasort. However, the improvement diminishes as the number of reduce tasks increases. This is because with multiple reduce tasks, the data is divided among them, and reducing the threshold may not have as significant an impact on parallelism. As the dataset size increases, the impact of *fetch_threshold* becomes more pronounced. Larger datasets benefit more from a lower *fetch_threshold* as they can take advantage of parallel processing early in the execution.

The results for wordcount experimentation are shown in figure 8. The impact of *fetch_threshold* observed with the wordcount benchmark is similar, although the overall execution time is significantly larger. Wordcount, being a CPU-intensive task, may see even greater improvements with lower *fetch_threshold* values, especially for larger dataset sizes. This is because CPU-bound tasks benefit more from increased parallelism.

This suggests that the impact of *fetch_threshold* on parallelism depends on the specific configuration and workload characteristics, highlighting the importance of optimizing this parameter based on the context of the application and cluster setup.

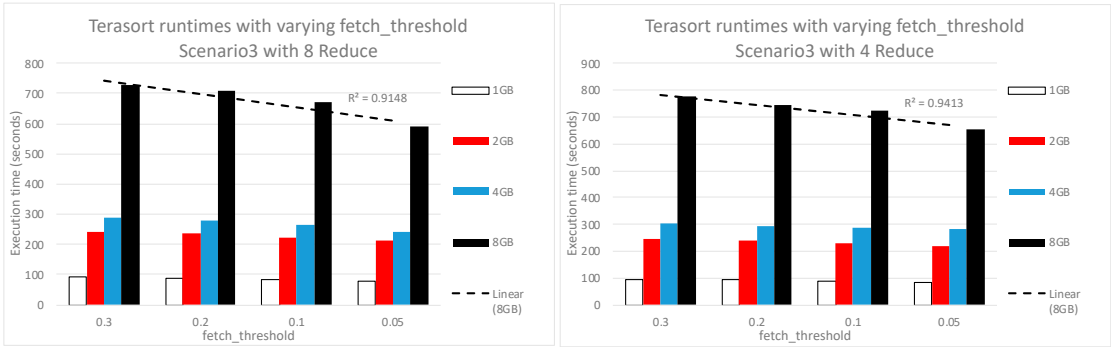


Figure 7. Comparison of *fetch_threshold* values for Terasort Execution times for Scenario3 with 8 and 4 reduce tasks with chunk sizes C=64MB.

 doi:10.20944/preprints202404.0154.v1

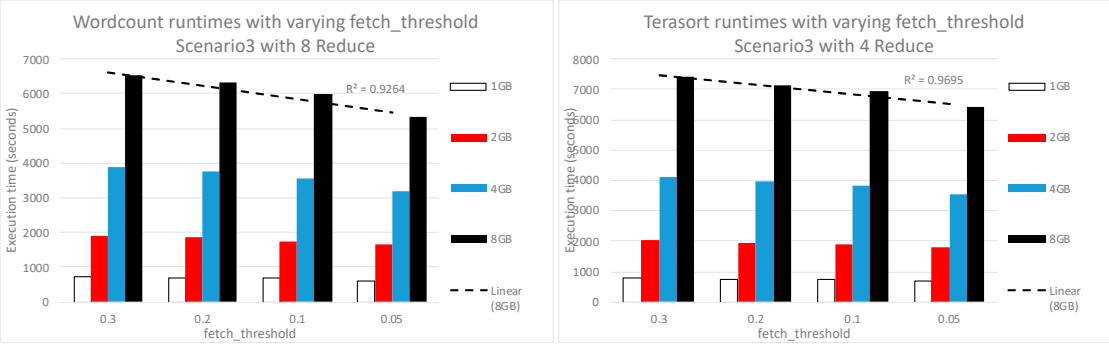


Figure 8. Comparison of *fetch_threshold* values for WordCount Execution times for Scenario3 with 8 and 4 reduce tasks with chunk sizes C=64MB.

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 etc., 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 SBCs, leading to the proposed changes in the YARN scheduling mechanism.

The proposed changes to the YARN scheduling mechanism aim to address the limitations posed by frugal SBC-based clusters. By introducing a *frugalityIndex* and *adaptiveConfig* policy, the redesign seeks to optimize resource allocation and enhance system efficiency. The *frugalityIndex* 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 *frugalityIndex* and *adaptiveConfig* policy into the YARN 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 *frugalityIndex* and *adaptiveConfig* policy, the study achieves better resource utilization and reduced task completion times. Furthermore, the impact of *fetch_threshold* on parallelism highlights the importance of fine-tuning parameters based on workload characteristics. Lower *fetch_threshold* values lead to improved parallelism, but the optimal threshold may vary depending on the specific configuration and workload.

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,

 doi:10.20944/preprints202404.0154.v1

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 SDCard storage media. We noted that the cluster required approximately 76W of power during wordcount execution. The overall power consumption ranged between 69W and 78W 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 Intel i7-12700KF @ 12-Core processor with 16GB RAM and a 500GB SSD. The power consumption for similar terasort and wordcount jobs ranged between 112W and 138W. We also noted that the task execution times on PC were 1.3x and 1.8x 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 a RPi5 is approx. 80 USD, it is reasonable to anticipate that the prices for these devices will 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. To this end, in this work we proposed changes in the YARN scheduling mechanism. By introducing a *frugalityIndex* and *adaptiveConfig* policy, the redesign seeks to optimize resource allocation and enhance system efficiency. The *frugalityIndex* serves as a crucial metric for categorizing SBC nodes based on their capabilities. By considering factors such as CPU speed and memory size, the index facilitates intelligent resource allocation, ensuring that tasks are assigned to nodes best suited to handle them. The *adaptiveConfig* policy enhances the flexibility of the YARN 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. The *fetch_threshold* for reduce tasks, enhances task prioritization and data processing efficiency.

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. The *frugal_conf* setting demonstrates significant performance improvements, executing 4.7 times faster than the *worst_native* scenario and 1.9 times faster than the *best_native* scenario. In terms of Terasort execution, Scenario3 outperforms Scenario1 by 5.86 times and Scenario2 by 1.79 times. Additionally, setting the *fetch_threshold* to 0.05 achieves 1.23 times faster runtimes for configurations involving 8 reduce tasks.

 doi:10.20944/preprints202404.0154.v1

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 *frugalityIndex* ensures efficient utilization of computational resources, further enhancing sustainability.

Acknowledgments: The author would like to thank Prince Sultan University for the payment of the Article Processing Changes.

References

1. Óscar Castellanos-Rodríguez, Roberto R. Expósito, Jonatan Enes, Guillermo L. Taboada, Juan Touriño, Serverless-like platform for container-based YARN clusters, *Future Generation Computer Systems*, Volume 155, **2024**, Pages 256-271, https://doi.org/10.1016/j.future.2024.02.013.
2. Warade, M.; Schneider, J.-G.; Lee, K. Measuring the Energy and Performance of Scientific Workflows on Low-Power Clusters. *Electronics* **2022**, 11, 1801. https://doi.org/10.3390/electronics11111801
3. 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. https://doi.org/10.3390/s24030970
4. 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. https://doi.org/10.1007/s11356-021-13248-3
5. A Setiyawan, 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 http://doi.org/10.1088/1742-6596/2406/1/012028
6. Ali, A.-e.A.; Mashhour, M.; Salama, A.S.; Shoitan, R.; Shaban, H. Development of an Intelligent Personal Assistant System Based on IoT for People with Disabilities. *Sustainability* **2023**, 15, 5166. https://doi.org/10.3390/su15065166
7. Netinant, P.; Utsanok, T.; Rukhiran, M.; Klongdee, S. Development and Assessment of Internet of Things-Driven Smart Home Security and Automation with Voice Commands. *Internet of Things* **2024**, 5, 79-99. https://doi.org/10.3390/iot5010005
8. Chen, I.-T.; Tsai, J.-M.; Chen, Y.-T.; Lee, C.-H. Lightweight Mutual Authentication for Healthcare IoT. **Sustainability 2022**, 14, 13411. https://doi.org/10.3390/su142013411
9. S.J.Johnston, P.J.Basford, C.S.Perkins, H.Herry, F.P.Tso, D.Pezaros, R. D. Mullins, E. Yoneki, S. J. Cox, J. Singer, Commodity single board computer clusters and their applications, *Future Generation Computer Systems* 89 (**2018**) 201–212. doi: https://doi.org/10.1016/j.future.2018.06.048
10. P. J. Basford, S. J. Johnston, C. S. Perkins, T. Garnock-Jones, F. P. Tso, D. Pezaros, R. D. Mullins, E. Yoneki, J. Singer, S. J. Cox, Performance analysis of single board computer clusters, *Future Generation Computer Systems* 102 (**2020**) 278–291. https://doi.org/10.1016/j.future.2019.07.040
11. B. Qureshi, A.Koubaa, On energy efficiency and performance evaluation of single board computer based clusters: A hadoop case study, *Electronics* 8 (2) (**2019**) 182. https://doi.org/10.3390/electronics8020182
12. E. Lee, H. Oh, D. Park, Big data processing on single board computer clusters: Exploring challenges and possibilities, *IEEE Access* 9 (**2021**) 142551–142565. http://doi.org/10.1109/ACCESS.2021.3120660
13. A.J.A.Neto, J.A.C.Neto, E.D.Moreno, The development of a low-cost big data cluster using apache hadoop and raspberry pi. a complete guide, *Computers and Electrical Engineering* 104 (**2022**) 108403. https://doi.org/10.1016/j.compeleceng.2022.108403
14. S Nugroho and A Widiyanto, Designing parallel computing using raspberry pi clusters for IoT servers on apache Hadoop, J. *Phys.: Conf. Ser.* 1517 012070 (**2020**), http://doi.org/10.1088/1742-6596/1517/1/012070
15. Apache Hadoop YARN, last accessed March 2024, https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html
16. A. Singh *et al*., "A Comparative Study of Bigdata Tools: Hadoop Vs Spark Vs Storm," *2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek)*, Kharkiv, Ukraine, **2023**, pp. 1-5, http://doi.org/10.1109/KhPIWeek61412.2023.10311577
17. J. Xue, T. Wang and P. Cai, "Towards Efficient Workflow Scheduling Over Yarn Cluster Using Deep Reinforcement Learning," *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, Kuala Lumpur, Malaysia, **2023**, pp. 473-478, http://doi.org/10.1109/GLOBECOM54140.2023.10436820

 doi:10.20944/preprints202404.0154.v1

18. B. Qureshi and A. Koubaa. On performance of commodity single board computer-based clusters: A big data perspective. *EAI/Springer Innovations in Communication and Computing*, **2020**, 349–375. https://doi.org/10.1007/978-3-030-13705-2_15
19. S. Vengadeswaran, S.R. Balasundaram, P. Dhavakumar, IDaPS — Improved data-locality aware data placement strategy based on Markov clustering to enhance MapReduce performance on Hadoop, *Journal of King Saud University - Computer and Information Sciences*, Volume 36, Issue 3, **2024**, 101973, https://doi.org/10.1016/j.jksuci.2024.101973.
20. Han, R., Liu, C. H., Zong, Z., Chen, L. Y., Liu, W., Wang, S., & Zhan, J. (**2019**). Workload-Adaptive Configuration Tuning for Hierarchical Cloud Schedulers. *IEEE Transactions on Parallel and Distributed Systems*, 30(12), 2879-2895. Article 8741093. https://doi.org/10.1109/TPDS.2019.2923197
21. N. Ahmed , Andre L. C. Barczak , Mohammad A. Rashid and Teo Susnjak1, A parallelization model for performance characterization of Spark Big Data jobs on Hadoop clusters, *Journal of Big Data* (**2021**) 8:107, https://doi.org/10.1186/s40537-021-00499-7