Computer Systems
Science and Engineering

Tech Science Press

DOI: 10.32604/csse.2023.036320

ARTICLE

# Performance Improvement through Novel Adaptive Node and Container Aware Scheduler with Resource Availability Control in Hadoop YARN

**J. S. Manjaly and T. Subbulakshmi***

School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, 600127, India

*Corresponding Author: T. Subbulakshmi. Email: subbulakshmi.t@vit.ac.in

Received: 26 September 2022 Accepted: 21 December 2022 Published: 09 November 2023

**ABSTRACT**

The default scheduler of Apache Hadoop demonstrates operational inefficiencies when connecting external sources and processing transformation jobs. This paper has proposed a novel scheduler for enhancement of the performance of the Hadoop Yet Another Resource Negotiator (YARN) scheduler, called the Adaptive Node and Container Aware Scheduler (ANACRAC), that aligns cluster resources to the demands of the applications in the real world. The approach performs to leverage the user-provided configurations as a unique design to apportion nodes, or containers within the nodes, to application thresholds. Additionally, it provides the flexibility to the applications for selecting and choosing which node's resources they want to manage and adds limits to prevent threshold breaches by adding additional jobs as needed. Node or container awareness can be utilized individually or in combination to increase efficiency. On top of this, the resource availability within the node and containers can also be investigated. This paper also focuses on the elasticity of the containers and self-adaptiveness depending on the job type. The results proved that 15%–20% performance improvement was achieved compared with the node and container awareness feature of the ANACRAC. It has been validated that this ANACRAC scheduler demonstrates a 70%–90% performance improvement compared with the default Fair scheduler. Experimental results also demonstrated the success of the enhancement and a performance improvement in the range of 60% to 200% when applications were connected with external interfaces and high workloads.

**KEYWORDS**

Big data; Hadoop; YARN; hadoop distributed file system (HDFS); MapReduce; scheduling; fair scheduler

## 1 Introduction

The advent of technology in the digital age has led to an explosion of data being created every nanosecond. This has accelerated the growth of new platforms that can work with these prolific data volumes, naturally called big data, due to its massive volume, velocity and variety of characteristics. In today's world, when massive amounts of data are widely available on easily-processed platforms, information has become the most effective asset influencing business strategy. Adopting information and insights to drive decisions in the business industry has led to the need for technology platforms with extensive computing capabilities and high performance. Several challenges distributed computing solutions had to overcome to meet increasing computing demands. The significant challenges of

traditional computing platforms are the absence of heterogeneous performance, inefficient resource management, unfair resource usage, and delayed performance. The Hadoop MapReduce framework [1] gained considerable popularity as compared to others because it has addressed the above-mentioned computing challenges. Apache Hadoop MapReduce was created by Doug Cutting and Mike Cafarella in April 2006, combining the existing ideas from the MapReduce white paper in 2004 [2] and the Google file system white paper published in 2003 by Google [3]. Splitting the entire process into chunks of map-reduce operations, moving code to data, and processing in a distributed fashion enabled Hadoop to solve many use cases in big data processing.

Apache Hadoop is widely accepted in the industry and is used in petabyte-scale workloads for structured and unstructured data processing in internet companies like Yahoo Inc. However, the first generation of the framework, Hadoop MapReduce v1, had many inherent resource management and scheduling limitations, leading to a few enhancements. The second generation of Hadoop, YARN [1,4], was created, which separated resource management and scheduling from application management, providing vast opportunities for performance improvement. In Hadoop YARN, an application is categorized into various individual tasks and executed in smaller execution units named containers. The YARN framework divides the application jobs into numerous tasks called "Map and Reduce" and distributes them to different worker nodes in the cluster, getting their execution accomplished in parallel. In contrast to Hadoop v1, where there was no dedicated resource management service, YARN resource management enabled Hadoop v2 to cater to applications with diverse resource needs. The scheduler in YARN identifies the application's resource requirements and allocates resources to individual tasks [5].

The First-In-First-Out (FIFO) Scheduler, the Fair Scheduler, and the Capacity Scheduler are the three default scheduling algorithms available in Apache Hadoop, which have been developed considering the out-of-the-box scheduling method. The Apache Hadoop default schedulers still presented challenges when connecting to external sources and processing enormous transformation-demanding workloads, despite considerable performance increases in cases where the data resided within the distributed cluster. Connecting to external systems that are either data sources or recipients is a common use case for Hadoop jobs in the real world of big data. The YARN schedulers, specifically the most popular Fair Scheduler [6], have limitations in connecting with many external connections and running applications simultaneously on the platform. Connection overload and frequent task breakdowns due to multiple concurrent external connections are among the most common problem spaces with such workloads. While various attempts have addressed this issue, the key challenge has been effectively improving the application performance when ingesting vast volumes of data from diverse application sources. A more adaptive and resource-aware scheduler would solve many common problems that arise as the orchestration of stages of the job can be tweaked mid-job, and the resources allocated to the job can be increased or decreased as the workload demands it. Node and container awareness also aids in distributing jobs in specific machines with external connectivity or the ability to host more Central Processing Unit (CPU) or Input Output (IO)-intensive jobs, especially in a heterogeneous environment. The node and container awareness feature allows an application to indicate the preferences for nodes and partitions in which the application intended to execute the tasks.

### *1.1 Innovation*

This paper has focused on eliminating the above-mentioned barriers to computational performance precisely where there is an ineffective utilization of containers and available resources. The main innovations of this research to improve the performance of Hadoop YARN are described below:

- It has developed a new scheduler, namely ANACRAC, for enhancing the performance of the Fair Scheduler, which will facilitate to scale of data science and data analytics applications running on massive amounts of data, taking into account the dynamic nature of workloads and the heterogeneity of Hadoop clusters.
- It has also emphasized the resource availability of the nodes and containers, which restrains tasks from being assigned to nodes and containers that have reached the specific threshold value.
- This paper has also demonstrated the adaptability of the container capacity and dynamically adjusting the application using the available CPU and memory.

### *1.2 Organization of the Article*

This paper is organized as follows: Section 2 explains the related work, and Section 3 describes the problem statement and the proposed model. The implementation details outlines in Section 4. Section 5 discusses the results and comparison with the existing work. The paper concludes with Section 6.

## 2 Literature Review

### *2.1 Review*

Hadoop is designed to process large amounts of data in parallel. The HDFS [1] splits files into blocks and distributes the files across a cluster of machines, along with a metadata file containing a list of other blocks that should be read from or written to. Hadoop minimizes network overhead by moving computation adjacent to the data, and multiple copies of this data block enable fault tolerance. The concept of data blocks enables the storage of huge data and processing efficiency through the MapReduce [1] algorithm. To improve the HDFS architecture and MapReduce framework's performance for massive data analytics, Hadoop schedulers were developed. Hadoop YARN reduces the challenges of the MapReduce scheduler by separating the ResourceManager from the Application Manager. Early approaches to schedules were, in general, made at system design time and were static. They failed to account for the dynamic scheduling [7] associated with operating systems where schedule changes are usually possible only in real-time. So, the underlying scheduling algorithm [8] determines how well the whole system can compute.

Schedulers were developed to satisfy the behaviour of a homogeneous environment in the past. Those previous schedulers were inattentive to the resource-specific constraints and outlines. In this regard, schedulers depend on policies to make task-assigning decisions. Resource capacity allocation [9] was grounded in the concepts of fairness, avoidance of halts, and allocation of adequate capacity, depending on pre-defined findings, to tasks in the queue. While the multi-tenant concept has been broadly adopted in this context, the premise for this efficiency expectation was that the data size was similar and that the compute resources were available throughout the cluster. The existing algorithms [10] can meet the requirements of data locality concerning two basic categories: space and time. When both of these conditions exist for a given set of data, they can exploit the locality of reference to improve the performance of the task without degrading its correctness. The real-life use cases of Hadoop computing are quite different from the ideal scenario. To be processed, data originate from sources external to the Hadoop environment. They might include various sources and formats, such as application logs, online services, transactions, pictures, and so on. Real-time data streaming, cloud-based applications, and heterogeneous environments bring a lot of complexity to the Hadoop environment to the parameters of space and time.

The edge node is a gateway connecting the Hadoop cluster with external systems and often forms the single point of failure depending on the quantum of factors involved. The ingestion speed is influenced by the number of hops and the amount of data that can be stored, making it unsuitable for enterprise-level streaming applications such as Netflix, Amazon prime, and Hotstar. Other factors influence the efficiency, such as the connectivity to external services for all the nodes. If this is limited due to security and other controls, the efficiency of existing schedulers within the cluster is impacted. Frequently occurring task breakdowns slow down the cluster substantially as the nodes need to reset the parameters enabling them to pick up from their broken state. Because of these things, a new scheduling policy [11] was needed to prevent and fix data ingestion problems and meet the computing needs of all kinds of applications without slowing them down. Distributed systems have traditionally centralized task breakdowns to achieve fairness between tasks. However, this does not equitably distribute resource [10,12] usage to nodes in the system, resulting in increased execution time and latency of certain tasks. The elimination of task breakdown necessitates the awareness of the resources available in individual nodes [13] and the number of slots or containers available [14]. Solving the above-mentioned problems also demands delayed scheduling based on the available containers [15].

Recent analytic applications demand the use of streaming information, computations, and computations for real-time data processing. High computing and real-time data processing have exposed that the former operational principles of fairness and deadline-based scheduling were insufficient to produce proportional uses in heterogeneous settings [16]. The high-consumption environment demands a highly efficient scheduler to get the best computing power and meet consumer expectations. This gap points out the need for resources such as the alignment of CPU and memory usage to the application's requirements. The former methods of scheduling in heterogeneous clusters and the cloud being oblivious of the machine-oriented specifications could not handle these benefits. Hence, there was an explicit requirement to assign application processing to the infrastructure facility of the cluster and accomplish the performance elevation through monitoring and controlling resource availability [17] and eradicating resource waste.

Numerous works have been performed to optimize big data. Yi et al. [18] have developed an adaptive Non-dominated Sorting Genetic Algorithm, the third version (NSGA-III), to optimize the big data efficiently and cost-effective. The performance of the NSGA-III can be improved by utilizing the operators such as Simulated Binary (SBX), Uniform Crossover (UC), and Single point (SI) crossovers [19]. Nath et al. [20] have created a novel decentralized Deep Deterministic Policy Gradient (DDPG) algorithm to use surrounding Multi-access Edge Computing (MEC) servers' collaboration to provide the best designs for multi-cell MEC systems. Simulation findings show that the suggested approach works better than other already used techniques, including Deep Q-Network (DQN). Bi et al. [21] have investigated the issue of determining the best offloading strategy to maximise the system utility for balancing throughput and fairness. The suggested algorithms can achieve effective performance in utility and accuracy, according to experimental findings. Several researches were conducted to develop an efficient scheduler framework to process big data effectively and solve the above-mentioned data. Zhang et al. [22] have presented a resource-aware MapReduce scheduler to handle this problem by segmenting job execution into three stages: processing, storage, and data transfer (network). An Adaptive Task Allocation Scheduler (ATAS) was created by Yang et al. [23] to boost the efficiency of Longest Approximate Time to End (LATE) schedulers in a heterogeneous cloud computing environment. To improve the backup task success rate, ATAS uses a technique to analyze the jobs that contributing to latency, compute the reaction time, and optimize the backup process. Mao et al. [24] offered a fine-grained dynamic MapReduce scheduling technique, which greatly reduces

task delay and maximizes resource efficiency. Each node's data, both past and present, is monitored so inefficient ones may be constantly identified and addressed.

A scheduler with elastic container configuration depends on real-time calculations [25] and improved efficiency on multi-tenant Hadoop backgrounds where repeated applications execute. A self-adaptive task tuning algorithm [13] has been accomplished to meet the resource adaptiveness [26] objective and achieve an exceeding performance from the heterogeneous cluster. Table 1 shows the summary of major existing works related to this study.

**Table 1:** Summary of the literature review

| Authors | Title of the article | Objectives | Findings |
|---|---|---|---|
| Hsin-Yu Shih; Jhih-Jia Huang; Jenq-Shiou Leu | Dynamic slot-based task scheduling based on node workload in a MapReduce computation model | To avoid underutilization of resources, this study proposes a slot-based task scheduling method that takes the physical burden on each node into account. | The evaluation results suggest that the proposed technique may improve the efficiency of computing across heterogeneous nodes in the cloud. |
| Norman Lim; Shikharesh Majumdar; Peter Ashwood-Smith | MapReduce constraint programming based resource management (MRCP-RM): A technique for resource allocation and scheduling of MapReduce jobs with deadlines | This paper focuses on the allocation of resources on the cloud and cluster-based framework by utilizing Mapreduce and Service Level Agreement (SLA). | The performance evaluation's findings provide insights into system behaviour and performance and show how well MRCP-RM/ Hadoop Constraint Programming based Resource Management (HCP-RM) performs in creating a schedule that results in a low percentage of tasks missing their deadlines (P). |
| Qi Zhang; Mohamed Faten Zhani; Yuke Yang; Raouf Boutaba David. R. Cheriton;Bernard Wong | PRISM: Fine-grained resource-aware scheduling for MapReduce | A phase-level scheduling technique has been developed, which increases execution parallelism and resource usage without creating stragglers. | This study ensured a significant optimization of resource utilization and 1.3 times improvement of performance. |

(Continued)

**Table 1 (continued)**

| Authors | Title of the article | Objectives | Findings |
|---|---|---|---|
| Jisha S. Manjaly; Varghese S. Chooralil | TaskTracker aware scheduling for Hadoop MapReduce | To develop a task tracker algorithm for improving the performance during computing in terms of working with data from external sources. | The task tracker algorithm improved the performance of MapReduce Hadoop. |
| Shivaswamy Rashmi, Anirban Basu | Resource-optimized workflow scheduling in Hadoop using stochastic hill climbing technique | The authors of this paper use a Stochastic hill climbing (SCH) soft computing technique to improve cloud workload, workflow response time, and resource utilization efficiency. | The SCH soft computing technique has improved the performance of the mentioned parameters. |
| Lauritz Thamsen; Benjamin Rabier; Florian Schmidt; Odej Kao | Scheduling recurring distributed dataflow jobs based on resource utilization and interference | This study describes a technique for scheduling recurrent data analysis tasks in shared cluster settings to maximize resource usage and work performance. | According to the results of their analysis, using an implementation based on Hadoop YARN may improve resource usage and reduce task runtimes. |
| Bin Ye; Xiaoshe Dong; Pengfei Zheng; Zhengdong Zhu; Qiang Liu; Zhe Wang | A delay scheduling algorithm based on history time in heterogeneous environments | The authors of this study suggest a novel scheduling method for a multi-user Hadoop cluster that incorporates the history time of the completed jobs and the strategy of the Delay scheduler. | The proposed algorithm has ensured a high performance while maintaining fairness in a shared heterogeneous environment. |
| Jiazhen Han; Zhengheng Yuan; Yiheng Han; Cheng Peng; Jing Liu; Guangli Li | An adaptive scheduling algorithm for heterogeneous Hadoop systems | To reduce the amount of tasks that are delayed, they created the CP-Scheduler (CPS) method, which employs an optimizer to evaluate the optimal schedule. | According to experimental findings, in scenarios of various sizes, the proportion of missed deadline work decreases by an average of 60%. |

(Continued)

**Table 1 (continued)**

| Authors | Title of the article | Objectives | Findings |
|---|---|---|---|
| Shin-JerYangYi-RuChen | Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds | To develop new innovative ATAS. | Their experimental results showed that ATAS can reduce 30% execution time. |
| Yingchi Mao; Haishi Zhong; Longbao Wang | A Fine-grained and dynamic MapReduce task scheduling scheme for the heterogeneous cloud environment | A Fine-Grained and dynamic MapReduce scheduling technique (FiGMR) is offered as a means to boost cluster performance in a heterogeneous cloud setting. | FiGMR provides higher map nodes to initiate backup map tasks. |
| Xiaoan Ding; Yi Liu; Depei Qian | JellyFish: Online performance tuning with adaptive configuration and elastic container in Hadoop yarn | This article suggests using JellyFish, an online performance tuning system, to boost the efficiency of MapReduce tasks and make better use of Hadoop YARN's available resources. | According to experimental findings, JellyFish can outperform default YARN in MapReduce task performance by an average of 65% for jobs performed repeatedly and 24% for jobs executed for the first time. |
| Dazhao Cheng; Jia Rao; Yanfei Guo; Changjun Jiang; Xiaobo Zhou | Improving performance of heterogeneous MapReduce clusters with adaptive task tuning | They introduced Ant, a self-adaptive task-tuning technique that searches automatically for the best settings for particular jobs across several nodes. | The average time it takes to do work is reduced by 31% according to experiments performed on a heterogeneous physical cluster with widely different hardware capabilities. |

### *2.2 Research Gap*

From the above-mentioned literature survey, it is evident that most authors have developed a framework to enhance the computing performance of the Hadoop platform by considering resource-aware, performance-aware, resource-aware and interference-aware approaches. The existing framework shows inefficient computing performance while connecting to external sources and processing enormous transformation-demanding workloads. The common problems of the existing framework

are connection overload and frequent task breakdowns. This study has developed an Adaptive Node and Container Aware Scheduler with a Resource Availability Control method to eliminate those issues and enhance computing performance. This research has focused on the node and container for resource allocation, which is unique from another developed framework. This paper proposes an improved version of the fair scheduler combining the node resource and container awareness with delay scheduling to eliminate task breakdowns. The proposed scheduler combining the elastic container behaviour and real-time self-adaptive task tuning ventured on functional containers put forward with resource consumption.

### *2.3 Step of the Model*

Fig. 1 represents the whole steps of this manuscript. This study started with a rigorous literature review to find the performance gap of Hadoop YARN.

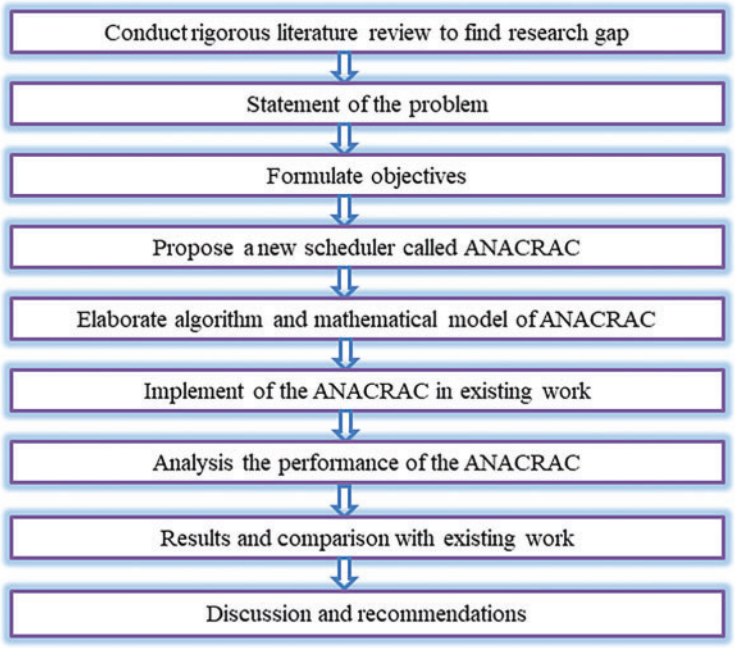


**Figure 1:** Workflow chart of steps of the manuscript

## 3 Problem Statement

The default schedulers of Apache Hadoop show significant workloads challenges during computing the data from external sources. Although the FIFO Scheduler, the Fair Scheduler, and the Capacity Scheduler are very effective in cases where the data resides within the distributed cluster, they show inefficient performance in connecting the data from external sources. One common issue with YARN is that its schedulers, especially the widely used Fair Scheduler, cannot handle too many simultaneous connections to external resources or processes. One of the most typical issues with such workloads is connection overload, which often leads to tasks failing due to having too many external connections open simultaneously.

### *3.1 Assumptions*

The proposed new scheduler aims to solve the drawbacks of Fair Scheduler's current implementation, especially in data ingestion applications dealing with external connectivity. The proposed scheduler ANACRAC, which the following three approaches to enhance the performance of the Fair Scheduler. ANACRAC builds on Fair Scheduler by introducing node-level control and container-aware partitioning (Section 3.1) with resource awareness (Section 3.2) and adaptiveness (Section 3.3). The scheduler was created by leveraging the Hadoop YARN's pluggable architecture to add new features and extend the default out-of-the-box schedulers. Table 2 represents the notations of the symbols of the equations.

**Table 2:** Notation of the symbol

| Symbol | Description |
|---|---|
| $A$ | Represent an application in a node |
| $N$ | The total set of the nodes in the cluster |
| $M$ | the set of eligible nodes for application |
| $i$ | Number of nodes |
| $j$ | Number of container |
| $AN_{ir}$ | The node restriction for the $i^{th}$ node for application A |
| $C$ | The number of containers present in a node |
| $P$ | The maximum number of concurrent containers per node |
| $AC_{ijr}$ | The container restriction for $j^{th}$ Container in the $i^{th}$ node for an application A |
| $UT$ | The time spent in user mode, |
| $NT$ | The time spent in nice mode |
| $ST$ | The time spent in system mode. |
| $LCT$ | The Last Cumulative CPU Time |
| $ST$ | Sample Time |
| $LST$ | The Last Sample Time |
| $NP$ | The Number of Processors |
| $DSS$ | Difference in Disk Status |
| $DS$ | The Disk Status |
| $LDS$ | The Last Disk Status |
| $SDST$ | The Sample Disk Status Time |
| $LSDST$ | The Last Sample Disk Status Time |
| $IOU$ | Input Output Usage |
| $N_iR$ | The total resource usage of application A in node j |
| $R_i$ | The resource usage of $i^{th}$ active container |
| $TR$ | Total resource usage of application A across the cluster |
| $AC_{it}$ | The new container allocation for application A at time "t" |
| $K$ | The default container |
| $R_s$ | The threshold value of the resource for adaptiveness |

### *3.2 Mathematical Model*

#### *3.2.1 Node and Container Awareness Mathematical Model*

The node and container awareness feature allows an application to indicate the preferences for nodes and partitions in which the application intends to execute the tasks. The framework will adhere to these preferences as much as possible and ultimately schedule these applications using the available resources. The default schedulers of a Hadoop cluster do not have any flexibility in choosing which resources to utilize. The node awareness feature significantly impacts applications that require shared contents or connections to external sharing nodes. One of the best candidates for node awareness is the Extract Transfer and Load (ETL) pipeline within data warehouses, where there is a need to connect with many external sources. This feature allows applications to be queued for the targeting node with the least running instances. These features aid in scheduling applications to best utilize the nodes. The ANACRAC scheduler's node awareness feature prevents tasks from running and failing if the NodeManager's load exceeds the application's threshold. This constraint eliminates the task breakdown at NodeManager by keeping the task limit under the threshold for the application.

Many existing schedulers used for scheduling nodes in large clusters dynamically allocate resources to application containers based on the applications' resource requirements and the overall cluster resource utilization [27,28]. However, those schedulers cannot effectively utilize resources if too many containers are within a single node. The container-awareness feature has been designed to allow the application to restrict the number of containers that can run concurrently in a node. This feature is provided by an overlay aware of the container information in the nodes. When an overlay receives a request to allocate a container, it checks the container allocation in its nodes. If the number of containers of the requested type on the node already exceeds the threshold, it refuses the request.

Node and container awareness scheduling algorithm distributes containers evenly across the YARN host. It will keep a new incoming task on hold if there is an unavailability of enough resources on the host. It also checks and validates the active containers and hosts before allocating the tasks to any active containers and nodes.

Node restriction can be calculated as follows:

$$AN_{ir} = \begin{cases} 1, & if\ i \in M \\ 0, & Otherwise \end{cases} \tag{1}$$

Container restriction can be calculated as follows:

$$AC_{ijr} = \begin{cases} 1, & if\ j \leq P \\ 0, & Otherwise \end{cases} \tag{2}$$

Node and container restriction together can be calculated as follows:

$$AN_{ir}C_{ijr} = \begin{cases} 1, & if\ i \in M,\ j \leq P \\ 0, & Otherwise \end{cases} \tag{3}$$

#### *3.2.2 Resource Awareness Mathematical Model*

The resource utilization ranges for the various frameworks supported by YARN depend on the application. Few applications require a large amount of CPU to process, and some others demand excessive IO resources. The data locality consideration in the Fair Scheduler, which is the parent scheduler of ANACRAC, leads to resource skewing and negatively impacts the Hadoop platform's performance. In addition, most organizations follow heterogeneous Hadoop cluster configurations

due to the on-demand addition of nodes to cater for the exponential growth of data analytics. This leads to irregularity in resource availability as different nodes are configured with diverse computing resources such as CPU, memory and Input-Output potentials.

To solve the performance limitations caused by resource availability, ANACRAC suggested the resource-awareness feature. A dynamic resource computing module proposed at the node level calculates the aggregated resource usage of the individual node at a point in terms of CPU and IO usage. The resource usages are calculated every second and sent back to the ResourceManager for further improvement of the ANACRAC scheduling algorithm.

Dynamic resource-awareness of the node supports scheduling of the applications with better performance of applications with high CPU or IO resource utilization. ANACRAC scheduler shows configuration parameters to list the application type, CPU/IO, and utilization threshold to limit the container allocation.

*CPU Usage*

Cumulative CPU Time (CCT) is computed as below:

$$CCT = UT + NT + ST \tag{4}$$

From Eq. (4), CPU usage '(CU)' can be computed as below:

$$CU = (CCT - LCT) * \frac{100F}{ST - LST} * NP \text{ [29]} \tag{5}$$

*IO Usage*

Difference in Disk Status (DSS) is computed as below:

$$DDS = DS - LDS \tag{6}$$

The Difference in Time is computed as below:

$$DT = SDST - LSDST \tag{7}$$

From Eqs. (6) and (7), Input Output Usage '(IOU)' can be computed as below:

$$IOU = \left((DDS) * \frac{100}{DT}\right) \text{ [29]} \tag{8}$$

### *3.2.3 Adaptive Scheduling*

Although applications are categorized and assigned containers depending on the resource controllability components, the resources may not be utilized entirely or over-utilized depending on the dynamic behaviour of applications and data. YARN schedulers assign resources to containers in a static manner that ignores actual application usage. ANACRAC has also focused on the resource utilization of individual nodes. As a result, containers of various applications running in the same node and resource usage do not denote the actual usage of the individual applications. CPU and memory are the configurable resources in the YARN containers. The resource requirements of containers in an application are computed at the early stage of application execution, and the Resource Manager allocates the necessary resources when required. Resource consumption of an application is dynamically calculated at a particular point in time, the resource utilization of active containers requires to be computing individually and grouping at the application level.

The application-level resource utilization can be computed on either the ApplicationMaster or the ResourceManager side. Resource awareness proposed a resource calculator module at the node level and broadcasted the calculated information to the ResourceManager. The adaptive scheduling reuses the architecture with an advanced version of container-level resource computation. Adaptive scheduling calculates the vCPU and physical memory usage of individual containers. This design points to leading the fine-grained container-level data back to the ResourceManager for extended computations. In addition to giving back information about resources, the scheduler looks at user configuration files to set the right values for a newly created container. Let 'N' is the total number of the nodes in the cluster. Each 'N' includes a different number of active containers for each application. Let 'm' is the total set of active containers for application A in $j^{th}$ Node $N_j$.

The total resource usage of application A in node j is denoted as "$N_jR$".

$$N_jR = \sum_{i=1}^{m} Ri \tag{9}$$

Total resource usage of application A across the cluster is denoted by "TR"

$$TR = \sum_{j=1}^{N} N_jR \tag{10}$$

The average resource usage of application A at time "t" is denoted as "$U_{at}$"

$$U_{at} = TR/M \tag{11}$$

The new container allocation for application A at time "t" is denoted as "$AC_{jt}$"

$$ACjt = \begin{cases} K, & if\ Uat \leq Rs \\ P, & Otherwise \end{cases} \tag{12}$$

### *3.3 Solution Approach*

#### *3.3.1 Proposed System Architecture*

The ANACRAC scheduler consists of four new modules that provide additional functionality to the Fair scheduler. These are the configuration loader, node check, container check and resource check. The configuration loader evaluates the eXtensible Markup Language (XML) configuration files that which evaluates the XML configuration files that are unique to the application during the application submission and updates the scheduler. Every time a new application is submitted, the scheduler looks for two configurations-the names of the nodes and the pre-defined threshold of concurrent containers which can run for the application. Fig. 2 shows the system architecture of the ANACRAC scheduler.

An application check module associated with the configuration loader is used to ensure compliance for applications on new request submissions. This module can read an attached file on submission, update its property using information from this XML configuration file, and update the NodeManager property on schedule. After scheduling, the NodeManager sends periodic status messages to ResourceManager explaining whether applications are compliant. Node check verifies that nodes are available by checking the scheduler module's node threshold property before allocating containers onto nodes. If both modules validate, ApplicationMaster allocates the container to the node that is running that application.

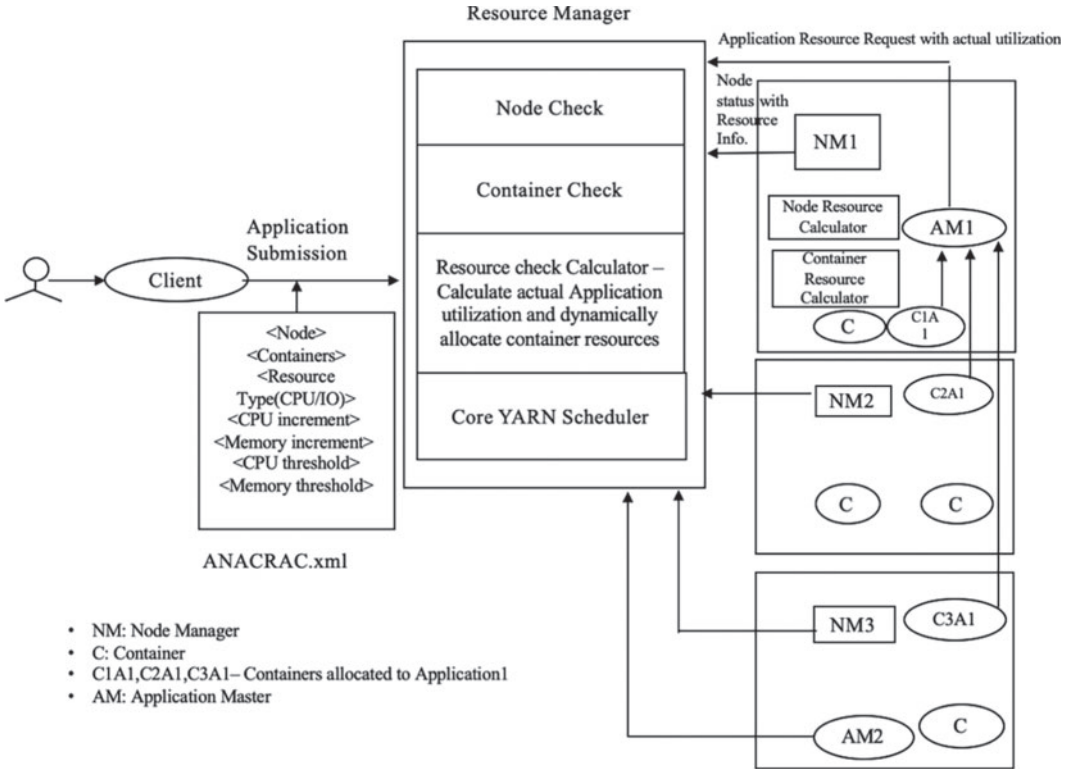


**Figure 2:** System architecture of ANACRAC

To calculate the typical CPU and IO use of individual nodes, a resource calculator module was developed. This component gathers node status data from the /proc/stat file and the iostat service and then calculates the resource use at the node level in each second and reports it to the ResourceManager. The YARN module employs the serialization standard known as Google protobuf. As defined and built by NodeResourceProto, its primary purpose is to serialize the dynamic resources' details before passing them on to ResourceManager from NodeManager. ANACRAC algorithm accepts the node resource information and verifies it with the user-submitted resource configuration. ANACRAC also lists the new tasks in the available containers of the node depending on the resource availability information delivered from the node. CPU and IO-intensive jobs are assigned to the node with the most CPU and IO. Adaptive scheduling is tied up with a resource check calculator module that computes the actual usage of application resources depending on the container resource usage. With the aid of Hadoop registry data, which examines the vCore and physical memory utilization, the container resource usages are calculated at each NodeManager. This container-level resource utilization is integrated into a map of active containers in the node and sent back to the ResourceManager aggregated with NodeStatus. To keep track of individual container status for inter-process communication, a new ContainerStatusProto is created. This resource information is verified with the threshold parameters offered by the user over the configuration file, and fine-tuning the container resource request depends on the actual usage. The scheduler assigns more resources to the newly spawned containers if the actual resource usage is high. So, adaptive scheduling improves the performance of a cluster by assigning resources based on how applications dynamically use resources.

### *3.3.2 Uniqueness of ANACRAC Scheduler*

- ANACRAC offers node-specific application submission and regulates the number of containers execute per node with the support of application-specific configuration parameters to improve the performance of data ingestion applications.
- ANACRAC introduces dynamic resource awareness along with node and container awareness for performance elevation of data analytics applications.
- ANACRAC manages data ingestion and analytic applications cooperatively in the homogeneous and heterogeneous environment, enhancing the scheduling algorithm using adaptive container resource allocation in the specific containers.

### *3.3.3 Algorithm of ANACRAC Scheduler*

The working process of the ANACRAC scheduler has been detailed in the below algorithm:

**Algorithm:** ANACRAC scheduler

**Input:** Heartbeat with container resource status received from the NodeManager
**Output:** Assigns the task to the node with updated container resource.
1. Initialize each application with "0" wait time and priority as "Normal"
2. Initialize application active containers registry with empty map
3. Read and set value of hostname, number of containers, application type, application threshold and container resource thresholds from application configuration
4. **If** heartbeat is received from the NodeManager **then**
5. Read the node status from the heartbeat
6. Read the hostname, running containers information, node resource report and container resource usage from node status
7. Update application's active containers registry with running containers resource usage from the Node
8. Calculate dynamic application resource usage from active container registry.
9. Sort applications by hierarchical scheduling policy with priority
10. **Repeat** until container allocation is done
11. Read application from the top of the sorted list
12. **If** the hostname of the NodeManager is belongs to the hostname list from configuration **then**
13. Filter the running containers of the application from the containers list
14. **If** the number of filtered containers less than the number of containers read from the configuration **then**
15. **If** node has a local container **then**
16. **If** the node resource usage is within the application threshold **then**
17. **If** application resource usage is within the threshold
Allocate container to the application and **return**
18. **Else** increment container resource and allocate container to the application **and return**
19. **Else** Log high resource usage and **return**
20. **Else If** priority is maximum **then**
21. Reset the waiting time as "0" and priority as "Normal”
22. Allocate the container to the application and **return**
23. **End If**

(Continued)

**Algorithm** (continued)

24. **End If**
25. **End If**
26. **If** the waiting time is greater than the maximum waiting time **then** set priority as "HIGH"
27. **Else** increment the waiting time by "1"
28. Remove the application from the list
29. **End If**
30. **End if**

## 4 Implementation of the Solution to Existing Project

The ANACRAC scheduler was implemented in Hadoop version 2.7.6. The pluggable architecture of YARN has enabled the scheduler to be apposite as a component for the hadoop-yarn-server-resource-manager module. ResourceManager is the master component, and NodeManagers act as the slaves in the YARN architecture. YARN implemented two-way communication between master and slave to handle the lifecycle of application and fault management. ResourceManager in scheduler is in charge of allocating resources to running applications based on the capacity and configuration constraints assigned to the nodes, whereas the NodeManager contains a set of containers, each of which can run one or more applications. The ResourceManager allows applications to use cluster resources following the constraints defined by the scheduler. ANACRAC updates the ResourceManager to implement the node and container awareness feature.

The scheduler updated two packages from the ResourceManager module to implement node and container awareness and their associated configuration changes. Those packages are built with the Eclipse IDE and Java 1.8, while RMAppImpl serves as the interface to an application in the ResourceManager. Then the updated class reads the new job configurations and transmits them to the scheduler to facilitate decision-making. The configurable components are all defined as XML properties, which can be converted to Java constants for application purposes. The scheduler implemented these new values to store attributes that were not previously specified, and the FairScheduler configuration class was created to house these new parameters. The original scheduling algorithm was implemented in a class called FSAppAttempt.

This study has evaluated the performance of the newly developed scheduler system. It has also compared the performance between the existing Hadoop schedulers and the newly developed ANACRAC scheduler. First of all, a twenty-node Hadoop YARN cluster was created in Amazon Web Service (AWS) [30] and configured with the scheduler. The node awareness functionality was tested by restricting connections to only 6 DataNodes (DataNode1-6), while the container awareness feature was tested by hard-wiring each node to a unique value. Table 3 shows the implementation details of ANACRAC scheduling systems.

**Table 3:** Implementation of ANACRAC

| Package | Class | Description |
|---|---|---|
| hadoop.yarn.server.ResourceManager.∗ | ResourceTrackerService | Retrieve container resource status from heartbeat message and set to ResourceManager context |
| | RMNode and RMNodeImpl | Getter and setter methods for container resource usage status of Node at Resource-Manager side |
| | RMNodeStatusEvent | Update container resource status to ResourceManager Node event. |
| | FairSchedulerConfiguration | Retrieve all configurations from XML file. |
| | RMAppImpl | Retrieve the application configuration object from configuration file |
| | FSAppAttempt | This is the main method which implemented ANACRAC algorithm. This class read all configurations from Application configuration object and set container resource based on adaptive resource calculations. |
| hadoop.yarn.util | LinuxResourceCalculatorPlugin | Calculate the current CPU and IO usage of the node. |
| hadoop.yarn.server.api.records.∗ | NodeResourceStatus | Class to hold node resource status information. |
| | NodeResourceStatusPBImpl | Google protobuf implementation of NodeResourceStatus for serialization. |

(Continued)

**Table 3 (continued)**

| Package | Class | Description |
|---|---|---|
| | ContainerStatusProto and ContainerStatusPBImpl | CPU and memory utilization of container in Google protobuf format. |
| hadoop.yarn.server.nodemanager | NodeStatusUpdaterImpl | Calculate the container resource usages from Hadoop Metrics registry and return a list of active containers with status. This class also calculate the overall resource usage of the node. |
| | ContainerImpl | Calculate the actual usage of CPU and physical memory in percent based on registry use and current allocation of container. |
| hadoop.yarn.server.util | BuilderUtils | Add container resource usage with existing container status object. |

The resource awareness feature was developed on the node and container awareness scheduler. Various modules were implemented for resource availability computation and data transfer between NodeManager and ResourceManager. The scheduler was assessed with the same configurations that were used for node and container awareness, and the results showed a significant increase in performance related to the former version.

The Resource-awareness capability of the scheduler was further enhanced with Adaptive scheduling to cater for the application's need to expand resources allocated dynamically for each stage of the process to achieve greater performance. Different classes and methods were implemented for calculating the resource consumption and availability in each container spawned and for the application-level aggregation of these container-level metrics. The scheduling algorithm was expanded with the supplementary modules, and the adaptive scheduler was developed as a pluggable scheduler for YARN. The scheduler was assessed with the same configurations used for other schedulers benchmarking, and the results showed a significant uptick in performance related to the former version.

ANACRAC scheduler is designed with a user configuration XML file that accepts the hostnames to run the job and schedule it on the configured nodes. The XML file supports the maximum number of concurrent containers in a node and resource requirements. The ANACRAC scheduler verifies an application's number of active containers in a node with the value received in the configuration. The scheduler restricts the container allocation if the number of active containers exceeds the configuration boundary. The two configuration properties for choosing nodes and containers can be set separately or

together with other properties based on resources in the node. The resource requirement configurations decide the behaviour of resource awareness and adaptive scheduling. Resource awareness properties can be used to define node-level CPU or IO usage thresholds. Adaptive scheduling properties can be used to control the adaptive scheduling behaviour based on the container-level resource usage. Table 4 specify the configuration properties that users can manage independently to control the application.

**Table 4:** Node and container awareness configuration details

| Configuration properties | Description |
|---|---|
| yarn.scheduler.anacrac.hosts | This property accepts the hostname of the nodes separated by a comma. Node and container awareness selects the nodes based on the value retrieved from this property. |
| yarn.scheduler.anacrac.tasks.maximum | This property sets the number of maximum concurrent containers runnable for an application in a node. |
| yarn.scheduler. anacrac.job.type | This property sets the type of the application. CPU, IO and ALL are the different job types available. |
| yarn.scheduler. anacrac.cpu.threshold | This property sets the threshold value of CPU usage of the node for the application. Accepted values-An integer value between 50-99. ANACRAC skip the task allocation if the calculated Node CPU usage goes beyond the limit. |
| yarn.scheduler.anacrac.io.threshold | This property sets the threshold value of IO usage of the Node for the Application. Accepted values-an integer value between 50-99. ANACRAC skip the task allocation if the calculated Node IO usage goes beyond the limit. |
| yarn.scheduler.anacrac.container.job.type | This property sets the resource type for Adaptive control of the application. CPU, Memory and ALL are the different job types available. |
| yarn.scheduler.anacrac.container.cpu.threshold | This property sets the threshold value of CPU usage of the application. Accepted values-Integer value between 50-99. ANACRAC increment the container vCPU if the calculated CPU usage goes beyond the limit. |

(Continued)

**Table 4 (continued)**

| Configuration properties | Description |
|---|---|
| yarn.scheduler.anacrac.container.memory.threshold | This property sets the threshold value of Memory usage of the application. Accepted values-Integer value between 50–99. ANACRAC increment the container physical memory allocation if the calculated memory usage goes beyond the limit. |
| yarn.scheduler.anacrac.container.cpu.increment | This property sets the CPU increment value for ANACRAC. Number of virtual cores allocation of container increment by this value. |
| yarn.scheduler.anacrac.container.memory.increment | This property sets the memory increment value for ANACRAC. Memory in Megabytes. |

## 5 Results and Comparison

To evaluate this scheduler's performance in heterogeneous and homogeneous clusters against different schedulers, many different sets of Hadoop applications such as SFTP file download, Sqoop import [31], TeraGen, TeraSort, Pi and Wordcount were set up in the environment. All these applications were used to test the performance in both types of clusters set up on the AWS public cloud.

To evaluate the effectiveness of the suggested scheduler, this study utilized a Sqoop import task, which transferred data from a single table in a relational database management system (RDBMS) to the HDFS. Sqoop imports data in parallel from the source database to HDFS. Characteristics of Sqoop import with a record size of up to 1.7 million rows assessed under multiple scheduler configurations. A distributed file download application used to evaluate the performance of external connections to the Hadoop cluster. The file download application uses the host machine's IP address, username, password, input path of files to download, output HDFS path and HDFS username as parameters. The file downloaded the job indexes the files specified in the remote input path and downloads the files into the HDFS path in a distributed way using the Secure File Transfer Protocol (SFTP). The SFTP file download job produces N number of containers where N was the number of files in the remote input path.

TeraGen takes as input the number of rows to be made and uses multiple mappers to make the rows of data simultaneously. The ideal number of mappers was the total number of vCPU-1 if TeraGen was the only application running in the cluster. Users could control the number of maps and reduce tasks by providing the configuration parameters while running the TeraSort application. By default, the number of mappers spawned in the Terasort is the number of input splits generated from TeraGen. The PI application can accept two parameters: the first parameter is the number of mappers, and the second is the number of samples per map. The wordcount application reads input text files and outputs the unique number of words and its occurrence in the entire dataset. Wordcount can be configured with n number of mappers where n can be configured as an input split of a single document; MapReduce split size or number of lines.

The ANACRAC scheduler was evaluated against the default Hadoop schedulers using different data sets. The scheduler was evaluated in 20 AWS Elastic Compute Cloud (EC2) machines with various types, container formats and node types. Evaluations were carried out on experimental traces for data ingestion workloads across different cluster topologies and sizes, along with multi-tenant clusters using the multi-tenant feature of Hadoop YARN. Three different machine types, namely t2.medium, t2.xlarge and t2.2xlarge, were utilized for testing. ANACRAC scheduler was assessed in the above environments with node and container awareness, resource awareness and adaptive resource awareness. Tables 5 and 6 explain the performance comparison of various improvements in ANACRAC against the Fair scheduler in the homogeneous and heterogeneous configurations in AWS environment. The node restriction is set to 6 and container restriction set to 2 for the evaluation purpose. All the values in the table are recorded in seconds.

**Table 5:** Comparison of fair, ANACRAC with NCA, RA and AS in public homogeneous cluster

| Scheduler | Mode | File Size in GB | WordCount | TeraGen | Pi | Sqoop import | TeraSort | SFTP down-load | Total time |
|---|---|---|---|---|---|---|---|---|---|
| Fair | N/A | 2.1 | 65 | 98 | 102 | 240(F) | 197 | 312(F) | 1014(F) |
| ANACRAC | NCA | | 63 | 84 | 74 | 21 | 163 | 202 | 607 |
| | RA | | 58 | 82 | 72 | 20 | 157 | 139 | 528 |
| | AS | | 55 | 74 | 67 | 18 | 133 | 119 | 466 |
| Fair | N/A | 6.4 | 72 | 124 | 112 | 377(F) | 222 | 412(F) | 1319(F) |
| ANACRAC | NCA | | 67 | 121 | 76 | 25 | 203 | 224 | 716 |
| | RA | | 65 | 78 | 62 | 22 | 144 | 169 | 540 |
| | AS | | 62 | 73 | 57 | 20 | 126 | 137 | 475 |

Note: NCA–Node and Container Awareness; RA–Resource Awareness; AS–Adaptive Scheduling; N/A–Not applicable; F–Failed status.

**Table 6:** Comparison of fair, ANACRAC with NCA, RA and AS in public heterogeneous cluster

| Scheduler | Mode | File Size in GB | WordCount | TeraGen | Pi | Sqoop import | TeraSort | SFTP down-load | Total time |
|---|---|---|---|---|---|---|---|---|---|
| Fair | N/A | 2.1 | 58 | 91 | 114 | 233(F) | 177 | 285(F) | 958(F) |
| ANACRAC | NCA | | 55 | 79 | 58 | 17 | 161 | 180 | 550 |
| | RA | | 53 | 62 | 59 | 14 | 138 | 128 | 454 |
| | AS | | 51 | 58 | 55 | 15 | 114 | 112 | 405 |
| Fair | N/A | 6.4 | 67 | 118 | 108 | 356(F) | 214 | 420(F) | 1283(F) |
| ANACRAC | NCA | | 63 | 113 | 70 | 18 | 206 | 204 | 674 |
| | RA | | 61 | 68 | 55 | 15 | 128 | 160 | 487 |
| | AS | | 60 | 60 | 54 | 14 | 112 | 121 | 421 |

Note: NCA–Node and Container Awareness; RA–Resource Awareness; AS–Adaptive Scheduling; N/A–Not applicable; F–Failed status.

Figs. 3 and 4 show the diagrammatic representation of Tables 5 and 6. Here, '*s*' indicates the time in seconds. Fig. 3 displays the performance of various Fair schedulers and ANACRAC in a

homogenous platform for different tasks such as WordCount, TeraGen, Sqoop Import, Terasort and SFTP download. From Fig. 3, it is evident that Fair scheduler has performed better than the proposed ANACRAC scheduler. When the data sources reside into the Fair Scheduler platform, the performance of the fair scheduler improves significantly. Fig. 4 represents the performance of Fair schedulers and ANACRAC in a heterogeneous platform for different tasks such as WordCount, TeraGen, Sqoop Import, Terasort and SFTP download. Now, the Fair scheduler shows less efficient performance than the proposed scheduler. When the data are supplied from external sources, the performance of the fair scheduler decreases significantly.

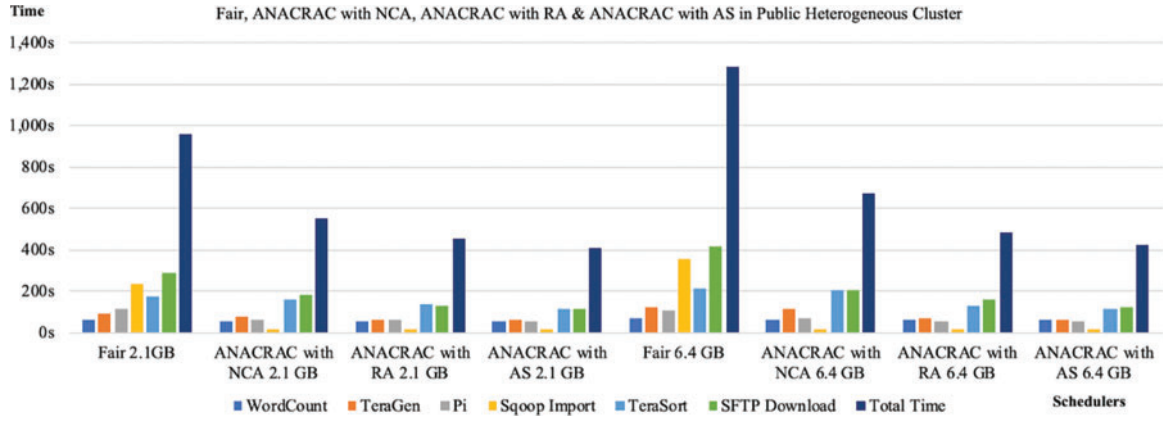


**Figure 3:** Comparison of fair, ANACRAC with NCA, RA and AS in public homogeneous cluster

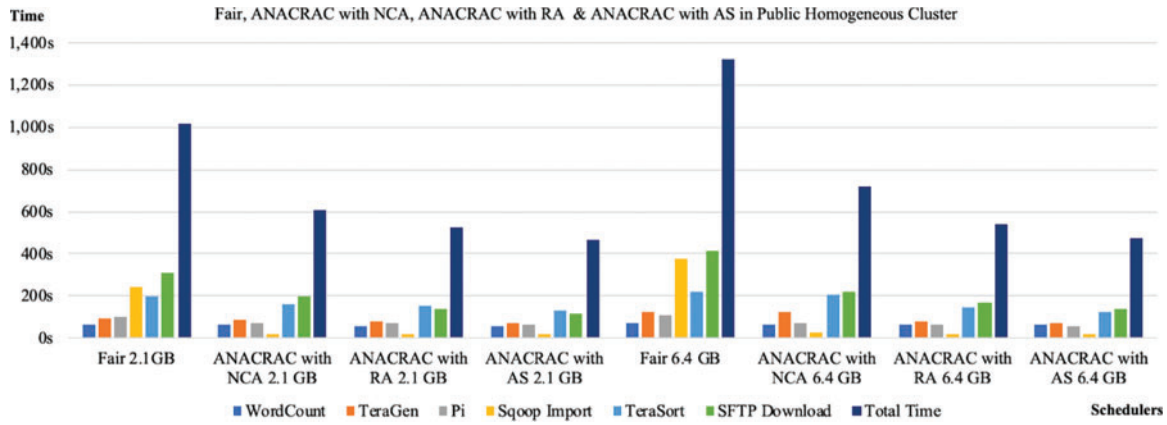


**Figure 4:** Comparison of fair, ANACRAC with NCA, RA and AS in public heterogeneous cluster

The Hadoop YARN implementation of the scheduler registered significant performance improvements in both homogeneous and heterogeneous clusters compared to the existing schedulers. It was noticed that the performance gain is better in large workloads for data ingestion applications. The SFTP file download with more than 2 GB datasets and Sqoop import from MySQL [32] with more than 300 K records proved a significant performance improvement in the testbed environment. Table 7 and Fig. 5 demonstrate the comparison study of FIFO, Fair, Capacity and ANACRAC schedulers against the testbed applications listed.

**Table 7:** Comparison of ANACRAC scheduler with Hadoop existing schedulers

| Scheduler | WordCount | TeraGen | Pi | Sqoop import | TeraSort | SFTP download | Total time |
|---|---|---|---|---|---|---|---|
| FIFO | 66 | 120 | 113 | 364(F) | 217 | 424(F) | 1304(F) |
| Fair | 64 | 116 | 110 | 327(F) | 214 | 418(F) | 1249(F) |
| Capacity | 67 | 118 | 108 | 356(F) | 209 | 420(F) | 1278(F) |
| ANACRAC | 60 | 60 | 54 | 14 | 112 | 121 | 421 |

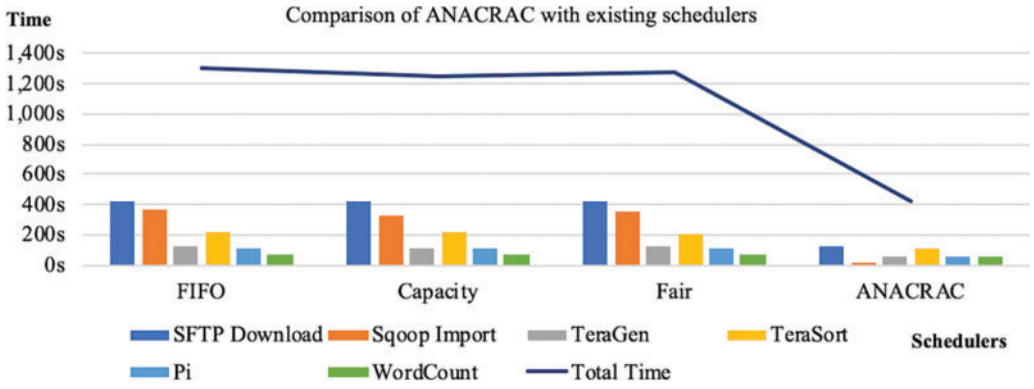


**Figure 5:** Comparison of ANACRAC scheduler with Hadoop existing schedulers

### *5.1 Comparison*

In Table 7 and Fig. 5, it is evident that the newly developed ANACRAC scheduler has shown the best performance. ANACRAC scheduler has taken less time than other schedulers like FIFO, Fair and Capacity. The existing scheduler in Hadoop YARN lowers the computational speed when the data are supplied from external sources. There is a lack of availability of resource management in the context of massive external data. When the data is located in internal sources, the existing scheduler of Hadoop YARN performs very well. The computational speed becomes high when the data are collected from internal sources. This study has developed a new scheduler system called ANACRAC which has eliminated the performance leggings of the Fair scheduler in terms of data from external sources. The newly developed ANACRAC scheduler has enhanced the performance of the Fair scheduler, which can facilitate to scale of data science and data analytics applications running on massive amounts of data, taking into account the dynamic nature of workloads and the heterogeneity of Hadoop clusters and data sources. The existing Fair Scheduler supports the batch processing of data, and it is not suitable for small data. Too many connections to external resources or processes at once are a typical problem for YARN's schedulers, notably the commonly used Fair scheduler. Connection overload is a common problem with such workloads; when there are too many external connections open at once, processes can't complete. This study has solved the issues of connection and slow computational speed by developing a novel ANRACRAC scheduler for the Hadoop YARN.

### *5.2 Discussion*

Node and container awareness was tested and evaluated with node and container restrictions. The results have validated that this ANACRAC scheduler demonstrates a 70%–90% performance

improvement compared with the default Fair scheduler. The performance exploration of the SFTP download and Sqoop import application for Fair scheduler and node and container awareness feature of the ANACRAC scheduler validates with 20 node homogenous and heterogeneous clusters with AWS EC2. Resource awareness offers resource availability of the specific nodes for scheduling. The resource awareness feature of the ANACRAC scheduler was assessed under the same AWS cloud. The results proved that 15%–20% performance improvement was achieved compared with the node and container awareness feature of the ANACRAC. The adaptive feature of the ANACRAC scheduler has been implemented and tested in the same cloud. The results are verified with both heterogeneous and homogeneous set-ups. The ANACRAC scheduler dynamically fine-tunes the freshly spawned containers depending on the run-time characteristics of the application. This dynamic fine-tuning improved the efficiency of the ANACRAC scheduler over the default schedulers. This scheduler showed a 30%–40% performance improvement compared to the resource awareness feature of ANACRAC.

The ANACRAC scheduler outperforms the default schedulers of the Hadoop package with varying input data sizes. The throughput and average latency improvement of the ANACRAC scheduler increased as the input data size. There was a noticeable speedup in performance with the increment of node numbers. It was also observed that there was a considerable reduction in the number of task breakdowns with ANACRAC scheduler implementation. After analysis of the results from Tables 5 and 6, the newly developed ANACRAC scheduler has shown more efficient performance than the existing schedulers. Though the parameters were the same for all schedulers, the ANACRAC scheduler has shown better performance than others. It is estimated that the ANACRAC scheduler has completed the task 60% to 200% faster than the previously developed scheduler. These percentages were calculated by comparing each scheduler system's required time of action in Tables 5 and 6.

## 6 Conclusion

The ANACRAC scheduler has been designed to enhance the performance of the Fair scheduler, the default scheduler provided by Apache Hadoop. The Fair scheduler cannot fulfil performance specifications for some applications due to an inability to allocate resources for applications dealing with external data sources or in situations of greater resource consumption. The scheduler's node and container awareness is a more flexible mechanism for applications, allowing them to request the resources they need in the cluster. Implementing newly developed ANACRAC scheduler enables Hadoop applications to tailor the resource requirements and helps to prevent resource breakdowns-thus optimizing utilization across the entire cluster. The scheduler's resource awareness offers application-level partitioning by leveraging application-specific information to prevent oversubscription of resources. The adaptiveness feature further enhances the performance of the scheduler using the container elasticity and dynamic resource consumption of the application at various stages of the runtime.

The major findings of this study are given below:

- The ANACRAC scheduler demonstrates a 70%–90% performance improvement compared with the default Fair scheduler.
- The results proved that 15%–20% performance improvement was achieved compared with the node and container awareness feature of the ANACRAC.
- The adaptive feature of the ANACRAC scheduler showed a 30%–40% performance improvement compared to the resource awareness feature of ANACRAC.

Overall, the ANACRAC scheduler achieves a performance improvement minimum of 60% to a maximum of 200% compared to the existing schedulers regarding data ingestion applications. The unique features of the ANACRAC scheduler make it a top pick for data ingestion use cases. Except the methods used in the paper, some of the most representative computational intelligence algorithms can be used to solve the problems, like monarch butterfly optimization (MBO), earthworm optimization algorithm (EWA), elephant herding optimization (EHO), moth search (MS) algorithm, Slime mould algorithm (SMA), hunger games search (HGS), Runge Kutta optimizer (RUN), colony predation algorithm (CPA), and Harris hawks optimization (HHO).

### *6.1 Managerial Insights and Practical Implications*

The suggested scheduler system will be very efficient to conduct big data analysis. The company which require to the analysis of a large amount of data can utilize this model to analyze their data efficiently. Data saved in HDFS for batch processing, stream processing, interactive processing, and graph processing may now be processed and performed with the aid of ANACRAC. Thus, it facilitates the operation of distributed applications apart from MapReduce.

### *6.2 Limitations and Further Study*

This work has some limitations regarding the effectiveness of the proposed framework ANACRAC. This study has evaluated the effectiveness of the ANACRAC using a single cloud service called Amazon Web Service (AWS) which may indicate less accurate results. This limitation can be eliminated by using multiple cloud services. Further study can be taken place by considering more external databases like oracle and TerraData. Further research can also be conducted by considering other new parameters to improve the efficiency of the newly developed algorithm.

**Acknowledgement:** With immense pleasure and a deep sense of gratitude, I wish to express my sincere thanks to my supervisor, who gave me advice on how to write the research paper and techniques in a professional manner. Also, I would want to express my gratitude to the university for providing me with infrastructural facilities and many other resources needed for my research.

**Funding Statement:** The authors received no specific funding for this study.

**Author Contributions:** The authors confirm contribution to the paper as follows: study conception and design: J. S. Manjaly; data collection: J. S. Manjaly; analysis and interpretation of results: J. S. Manjaly; draft manuscript preparation: J. S. Manjaly, T. Subbulakshmi. Both authors reviewed the results and approved final version of the manuscript.

**Availability of Data and Materials:** The data used to support the findings of this study are included in the article.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

[1] T. White, "Hadoop, HDFS, YARN & MapReduce," in *Hadoop: The Definitive Guide*, 4th ed., Sebastopol, CA, USA: O'Reilly Media, pp. 23–112, 2015.

[2] D. Jeffrey and G. Sanjay, "MapReduce: Simplified data processing on large clusters," *Communications of the Association for Computing Machinery*, vol. 51, no. 1, pp. 107–113, 2008.

[3] G. Sanjay, G. Howard and L. Shun-Tak, "The Google file system," *Association for Computing Machinery Special Interest Group in Operating Systems*, vol. 37, no. 5, pp. 29–43, 2003.
[4] Apache Hadoop YARN: Apache Hadoop, 2018. [Online]. Available: https://hadoop.apache.org/docs/r2.7.6/hadoop-yarn/hadoop-yarn-site/YARN.html
[5] Z. Wen, C. Zhang, J. Wu and J. Mo, "Research on mixed tasks scheduling in YARN," in *Proc. ITOEC*, Chongqing, China, pp. 226–230, 2017.
[6] Hadoop Fair Scheduler: Apache Hadoop, 2016. [Online]. Available: https://hadoop.apache.org/docs/r2.7.3/hadoop-yarn/hadoop-yarn-site/FairScheduler.html
[7] H. Shih, J. Huang and J. Leu, "Dynamic slot-based task scheduling based on node workload in a MapReduce computation model," in *Anti-Counterfeiting, Security, and Identification*, Taipei, Taiwan, pp. 1–5, 2012.
[8] R. Johannessen, A. Yazidi and B. Feng, "Hadoop MapReduce scheduling paradigms," in *Proc. of ICCCBDA*, Chengdu, China, pp. 175–179, 2017.
[9] N. Lim, S. Majumdar and P. Ashwood-Smith, "MRCP-RM: A technique for resource allocation and scheduling of MapReduce jobs with deadlines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1375–1389, 2017.
[10] M. Zhang, F. Zhani, Y. Yang, R. Boutaba and B. Wong, "PRISM: Fine-grained resource-aware scheduling for MapReduce," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 182–194, 2015.
[11] J. S. Manjaly and V. S. Chooralil, "TaskTracker aware scheduling for Hadoop MapReduce," in *Proc. of ICACC*, Cochin, India, pp. 278–281, 2013.
[12] S. Rashmi and A. Basu, "Resource optimized workflow scheduling in Hadoop using stochastic hill climbing technique," *IET Software*, vol. 11, no. 5, pp. 239–244, 2017.
[13] G. K. Archana and V. D. Chakravarthy, "HPCA: A node selection and scheduling method for Hadoop MapReduce," in *Proc. of ICCCT*, Chennai, India, pp. 368–372, 2015.
[14] L. Thamsen, B. Rabier, F. Schmidt, T. Renner and O. Kao, "Scheduling recurring distributed dataflow bobs based on resource utilization and interference," in *Proc. of Big Data Congress*, Honolulu, USA, pp. 145–152, 2017.
[15] B. Ye, X. Dong, P. Zheng, Z. Zhu, Q. Liu *et al.,* "A delay scheduling algorithm based on history time in heterogeneous environments," in *Proc. of China Grid Annual Conf.*, Los Alamitos, CA, USA, pp. 86–91, 2013.
[16] J. Han, Z. Yuan, Y. Han, C. Peng, J. Liu *et al.,* "An adaptive scheduling algorithm for heterogeneous Hadoop systems," in *Proc. of ICIS*, Wuhan, China, pp. 845–850, 2017.
[17] A. V. Panicker and G. Jisha, "Resource aware scheduler for heterogeneous workload based on estimated task processing time," in *Proc. of ICCC*, Trivandrum, KL, IN, pp. 701–704, 2015.
[18] J. Yi, S. Deb, J. Dong, A. Alavi and G. Wang, "An improved NSGA-III algorithm with adaptive mutation operator for Big Data optimization problems," *Future Generation Computer Systems*, vol. 88, no. 2, pp. 571–585, 2018.
[19] J. Yi, L. Xing, G. wang, J. Dong, A. Vasilakos *et al.,* "Behavior of crossover operators in NSGA-III for large-scale optimization problems," *Information Sciences*, vol. 509, no. 15, pp. 470–487,2020, 2020.
[20] S. Nath and J. Wu, "Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems," *Intelligent and Converged Networks*, vol. 1, no. 2, pp. 181–198, 2020.
[21] R. Bi, Q. Liu, J. Ren and G. Tan, "Utility aware offloading for mobile-edge computing," *Tsinghua Science and Technology*, vol. 2, no. 2, pp. 239–250, 2020.
[22] Q. Zhang, M. F. Zhani, Y. Yang, R. Boutaba and B. Wong, "PRISM: Fine-grained resource-aware scheduling for MapReduce," *Institute of Electrical and Electronics Engineers Transactions on Cloud Computing*, vol. 3, no. 2, pp. 182–194, 2015.
[23] S. J. Yang and Y. R. Chen, "Design adaptive task allocation scheduler to improve MapReduce performance in heterogeneous clouds," *Journal of Network and Computer Applications*, vol. 57, no. 4, pp. 61–70, 2015.

[24] Y. Mao, H. Zhong and L. Wang, "A fine-grained and dynamic MapReduce task scheduling scheme for the heterogeneous cloud environment," in *Proc. of DCABES*, Guiyang, China, pp. 155–158, 2015.
[25] X. Ding, Y. Liu and D. Qian, "JellyFish: Online performance tuning with adaptive configuration and elastic container in Hadoop Yarn," in *Proc. of ICPADS*, Melbourne, Australia, pp. 831–836, 2015.
[26] D. Cheng, J. Rao, Y. Guo, C. Jiang and X. Zhou, "Improving performance of heterogeneous MapReduce clusters with adaptive task tuning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 774–786, 2017.
[27] R. Lotfi, Y. Mehrjerdi, M. Pishvaee, A. Sadeghieh and G. Weber, "A robust optimization model for sustainable and resilient closed-loop supply chain network design considering conditional value at risk," *Numerical Algebra Control and Optimization*, vol. 11, no. 2, pp. 221–253, 2021.
[28] Y. Mehrjerdi and R. Lotfi, "Development of a mathematical model for sustainable closed-loop supply chain with efficiency and resilience systematic framework," *International Journal of Supply and Operations Management*, vol. 6, no. 4, pp. 360–388, 2019.
[29] Linux manual page: Linux, 2021. [Online]. Available: https://www.man7.org/linux/man-pages/man1/man.1.html
[30] Cloud Services-Amazon Web Services (AWS): Amazon Web Services, 2022. [Online]. Available: http://aws.amazon.com
[31] Sqoop User Guide (v1.4.2): Apache Sqoop, 2021. [Online]. Available: https://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html#_purpose
[32] MySQL Employees Sample Database: MySQL, 2022. [Online]. Available: https://dev.mysql.com/doc/employee/en