Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606

Volume 07 Issue 02
2024

Research Article
https://doi.org/10.56979/702/2024

Dynamic Load Balancing and Task Scheduling Optimization in Hadoop Clusters

Haiqa Mansoor[1], Bilal Aslam[1], and Usman Akhtar[2*]

[1]Riphah School of Computing and Innovation, Riphah International University Lahore, Lahore, Pakistan.
[2]Berlin School of Business and Innovation GmbH, Berlin, 12043, Germany.
[*]Corresponding Author: Usman Akhtar. Email: usman.akhtar@berlinsbi.com

Received: March 11, 2024 **Accepted:** June 22, 2024 **Published:** September 01, 2024

Abstract: Hadoop is a widely utilized distributed file system and processing framework for handling large-scale data. Nonetheless, the inherent load balancing and task scheduling mechanisms in Hadoop exhibit inefficiencies that may result in performance bottlenecks. In this paper, we propose a novel dynamic load-balancing algorithm designed specifically for Hadoop clusters. Our algorithm continuously monitors the performance indicators of nodes and dynamically adjusts task-node allocations to ensure equitable load distribution within the cluster. Furthermore, we consider the execution states of tasks to optimize resource allocation effectively. The primary contribution of this study resides in the analysis and resolution of load balancing and scheduling issues within Hadoop. In addition, our proposed dynamic scheduling algorithm also accounts for task execution states, thereby facilitating optimized resource allocation. We validate our algorithm across various workloads, demonstrating that it surpasses existing methods in job completion time, scalability, and resource utilization. The findings indicate that the proposed algorithm efficiently balances cluster loads, expedites task completion, and reduces both costs and resource consumption.

Keywords: Cloud; Load Balancing; Hadoop; Optimized Load Balancing; Hadoop Cluster.

1. Introduction

With the advancement of the Internet and the demand for data, many cloud service providers are offering services to their customers. Due to the need for horizontal scaling of storage in the era of Big Data, the Hadoop Distributed File System (HDFS) has attracted the attention of many researchers. Hadoop is the most widely used and effective solution for storing and processing large amounts of data. To process large amounts of data in its cluster, Hadoop uses the Hadoop Distributed File System and the MapReduce model platform. This has made high-performance load balancing an important consideration in ensuring the availability and performance of these applications. Hadoop is the most widely used and effective solution for storing and processing large amounts of data. Hadoop uses the Hadoop Distributed File System (HDFS) and the MapReduce programming model to process large amounts of data in its cluster [1].

Hadoop is a popular open-source framework for storing and processing large data sets. It is used by companies such as Yahoo, Facebook and Amazon. The core components of Hadoop are the Hadoop Distributed File System (HDFS) and MapReduce. HDFS provides reliable storage and data management, while MapReduce processes data in the cluster in parallel [2]. Hadoop breaks a job into smaller tasks, called map tasks, and reduce tasks. Map tasks read data from HDFS, split it into smaller pieces, and perform some computation on each piece. Reduce tasks and then combine the results of the map tasks [3]. Hadoop's distributed architecture makes it efficient for writing, reading, and processing data. It is a popular choice for a variety of big data applications, including data warehouse, data mining, and machine learning.

The HDFS is an open-source distributed file system designed to run on commodity hardware. It is a scalable, fault-tolerant file system that can be used to store large datasets.

HDFS was originally developed by the Apache project, but it has since become a popular choice for a variety of big data applications. HDFS uses a master-slave architecture. The master node, called the

NameNode, manages the file system namespace and metadata. The slave nodes, called *DataNodes*, store the actual data blocks [4]. HDFS is designed to be fault tolerant. If a *DataNode* fails, the *NameNode* will reassign the blocks that were stored on that node to other *DataNodes*. This ensures that the data is always available. Orchestration and preservation of the file system, file namespace, and metadata are within the purview of the name node. This central node keeps an inventory of the quantity of data nodes within the Hadoop infrastructure and their corresponding file associations. Data nodes, on the other hand, execute storage actions as directed by the name node and periodically provide updates to the *NameNode* about the blocks they are hosting [4]. HDFS is a powerful tool for storing and processing large datasets. It is used by a variety of organizations, including Google, Facebook, and Amazon. HDFS comprises a primary component known as the name node, acting as the master, alongside multiple data nodes that serve as slave nodes.

The load balancing problem has received a lot of attention and research in recent years. When a considerable volume of tasks is executed on a singular node, load balancing is achieved by redistributing tasks from overloaded nodes to those with less workloads. This strategic shifting of tasks results in a reduction in the overall duration of task execution, contributing to enhanced efficiency. In Hadoop, the Yet another Resource Negotiator (YARN) framework manages load balancing. Task scheduling is exclusively handled by the scheduler, aligning tasks with the available resources.

YARN offers three default scheduler types of FIFO, capacity, and fair schedulers each catering to distinct resource allocation scenarios. FIFO (First in First Out) algorithm used in the legacy version of the Apache Hadoop framework. When the jobs are submitted to the queue, the master node selects the job in order of submission. The most straightforward scheduler within YARN is the FIFO scheduler, which operates without the need for any configuration. Meanwhile, the capacity scheduler sets aside a distinct queue for small jobs, enabling swift initiation upon request. However, this approach comes with a trade-off, as it divides the cluster's capacity [5]. Consequently, larger jobs encounter prolonged completion times due to resource allocation. In contrast, the fair scheduler eliminates the need for capacity reservations. Dynamically distributes resources among all approved jobs. The main objective of the Fair scheduler is to share the resources between the tasks and to make sure that the data are available to the next available resource. Here, jobs are group into groups and this algorithm improves the allocation of data locality and provides fast response times in a shared Hadoop cluster [6]. The priority of the job is considered. The applications of YARN can share the resources of the huge Hadoop cluster using the Fair Scheduler, and these resources are dynamically managed so that no prior capacity is required. Resources are allocated so that each application in a cluster gets the same amount of time. Fair Scheduler makes scheduling decisions based on memory but can also be configured to work with CPU.

Several researchers have made attempts to tackle load balance and job assignment in Hadoop. One such approach is the Constraint-Based Hadoop Scheduler, as outlined in a recent study by Kumar et al. [7]. However, it is important to note that this method may not be entirely practical considering the inherent heterogeneity of Hadoop clusters, where job tasks may vary in completion time when executed on different data nodes. Tasks within Hadoop are allocated based on data location; in simpler terms, they are assigned to nodes storing the respective data.

In this research, we aim to address critical issues pertaining to Hadoop, load balancing and task scheduling. These aspects play a pivotal role in shaping Hadoop's overall performance. Our primary contribution lies in the examination and resolution of load balance and scheduling challenges within Hadoop. Effective load balancing stands to significantly enhance the efficiency of Hadoop, impacting task execution times and, in turn, job completion. Additionally, our proposed dynamic scheduling algorithm considers the execution state of tasks, enabling optimized resource allocation. This algorithm represents a substantial advancement over existing approaches by achieving dynamic load balancing, which allocates jobs based on task priorities and node capabilities. This innovation is poised to elevate Hadoop's capabilities across various dimensions, ultimately leading to improved overall performance.

The structure of the paper is as follows. Section 2 offers a comprehensive review of recent advancements in cloud load balancing. In Section 3, we delve into the detailed description of our proposed architecture. Moving forward, Section 4 is dedicated to the analysis and presentation of our research findings. Finally, Section 5 presents the conclusive remarks and key takeaways from this study.

2. Related Work

Scholars have dedicated considerable research efforts to optimizing data allocation for load balancing. Our literature review begins with a comprehensive review of load balancing across diverse computing domains, encompassing cloud computing, grid computing, and wireless networks, before focusing on the specific domain of Hadoop load balancing. Various load-balancing algorithms have been proposed, including well-known methods [8–10] such as round-robin, weighted round-robin, least-connection, weighted least-connection, and shortest expected delay. In the area of load balancing, the major contribution comes from the introduction of the Central Load-Balancing Decision Model (CLBDM) [11]. This novel approach extends the widely recognized round-robin load-balancing algorithm, introducing a layer of session switching at the application level. The CLBDM operates by meticulously monitoring the connection establishment duration between clients and nodes. Notably, this algorithm incorporates a critical threshold mechanism; should the connection time surpass this threshold, the connection is promptly terminated. Subsequently, the responsibility originally assigned to the node is seamlessly transferred to another node in the system. This dynamic approach to load balancing enhances the efficiency and reliability of resource allocation in distributed systems [8]. Their innovative approach draws inspiration from the coordinated behavior of ants to collect data from cloud nodes and efficiently assign tasks to specific nodes. Nonetheless, their method grapples with synchronization challenges, a concern that has subsequently been addressed [9].

In the field of load balancing research, Ni et al. (2011) [12] present an intriguing approach in their study. Their work focuses on designing a virtual machine mapping policy for private cloud environments, emphasizing multi-resource load balancing. In this policy, central scheduling controller assesses the availability of resources specific to a given job and subsequently determines task assignments. Complementing this controller, a resource monitor continuously gathers critical information pertaining to resource availability within their algorithm. This integrated approach ensures efficient resource allocation within private cloud infrastructures. Furthermore, T. Wu et al. have made significant strides in addressing the challenge of system stress resulting from data duplication and redundancy [10]. Their innovative solution introduces the Index Name Server (INS), a novel architecture designed for email-to-phone (SIP) management within data centers. The INS system incorporates deduplication and access point selection optimization techniques, effectively mitigating data redundancy issues. In the context of load balancing, the INS dynamically monitors key parameters, such as IP information and the busy level index, ensuring that load distribution remains balanced across the data center infrastructure. This approach enhances the overall efficiency and reliability of data management within data centers.

In another notable contribution, S. W. et al. present a load balancing methodology that combines elements from Opportunistic Load Balancing (OLB) and Min-Min Load Balancing (LBMM) algorithms [13]. OLB, though an established static load balancing technique, exhibits shortcomings in its treatment of node execution times, often leading to slower task processing. To mitigate this issue, the proposed approach introduces a sophisticated three- layered design and systematically partitions tasks into multiple subtasks. This innovative approach enhances task processing speed while maintaining effective load balancing, contributing to improved overall system performance.

Several notable contributions have advanced the field of load balancing in Hadoop. The Active Monitoring Load Balancing (AMLB) approach, as proposed by Mahalle et al. [14], prioritizes assigning requests to virtual machines with the lowest load. Another modified algorithm as a locally optimized load balancing solution is proposed by S. G. Domanal et al. [15]. The algorithm aims to equitably distribute incoming jobs among servers or virtual machines. It employs a directory counter with a virtual machine slope in addition to their status, enabling efficient virtual machined selection in the Load Balancer. However, a limitation lies in the algorithm's oversight of the present machine load, which may affect load distribution efficiency. To overcome this limitation, J. Adhikari and S. Patil [16] introduced the Double Threshold Energy-Aware Load Balancing (DT-PALB) method. The program manages to compute nodes depending on utilization percentages and reduces energy consumption by simultaneously turning off idle compute nodes.

Within Hadoop, scheduling and allocation decisions operate at the task and node level, applying to both the map and reduce phases. This means that not all tasks within a job are necessarily scheduled simultaneously. J. Gautam et al. [17] have categorized Hadoop schedulers based on factors including time and priority. Static scheduling, which predetermines job-to-node assignments before execution, aims to minimize overall job execution time. Dynamic scheduling, on the other hand, assigns jobs to nodes as they are being executed, optimizing resource utilization. Scheduling strategies based on resource availability aim to enhance Hadoop's performance by efficiently harnessing CPU, storage, and memory resources. Time-based scheduling centers around job deadlines, determining whether a job can be completed within a specific timeframe and how resources should be allocated accordingly [17].

Several Hadoop and MapReduce scheduling algorithms are designed around factors like data locality, resource availability, and performance. In their work, M. Hammoud et al [18], they introduce the Center-of-Gravity Reduction Scheduler (CoGRS). CoGRS minimizes MapReduce network traffic by incorporating locality and skew awareness in reduced task scheduling decisions, ensuring that each reduced job runs on the node with the highest center of gravity. Kumar et al. [19] introduced the Context-Aware Scheduler for Hadoop (CASH), a scheduler designed to adapt to the heterogeneity of Hadoop clusters. CASH demonstrates an understanding of the cluster's diversity and incorporates context learning, which encompasses job and resource characteristics. The CASH algorithm categorizes jobs based on their CPU and I/O demands and, in parallel, categorizes nodes according to their capabilities. Subsequently, it schedules jobs by considering contextual information, resulting in efficient and tailored resource allocation.

A resource and deadline-aware Hadoop job scheduler is proposed in Zhang et al. [20] suggest a next-knode scheduling (NKS) technique, prioritizing map tasks with strong data locality to improve node utilization. Y. Zhao et al. [21] introduce the Distributed Data Warehouse-based Work Scheduling Algorithm (TDWS), categorizing jobs by type and employing memory-aware techniques for more efficient scheduling. M. Yong et al. [22] propose a resource-aware scheduler that monitors node resource loads and matches jobs to their specific resource requirements. R. Nanduri et al. [23] develop a job-aware MapReduce scheduling algorithm utilizing heuristics and machine learning to select jobs compatible with available resources. H. Mao et al. [24] present a load-driven task scheduler featuring adaptive DSC, which dynamically assigns tasks based on data node workload and adjusts slot allocation. Y. Li et al. [25] offer a MapReduce scheduler and a power-aware rescheduling technique for a heterogeneous environment. The power-aware rescheduling strategy considers how to save energy while processing jobs and storing data.

To the best of our knowledge, only a limited body of work has been presented [26]. In their contribution, the authors introduce a scheduler that is distinguished by its dependence on constraints and conditions. This scheduler incorporates user-specified deadlines into a job-execution cost model as input parameters. Notably, it provides users with prompt feedback regarding the feasibility of completing a project within a given deadline, even allowing for deadline adjustments. By efficiently allocating the maximum number of open slots to the current running job, this model optimizes job execution within a Hadoop cluster. Additionally, this scheduler introduces a preemptive mechanism designed to reallocate resources from jobs with durations shorter than the required time limit [26].

In summary, the literature review exposes a wide spectrum of scheduling techniques and resource allocation strategies in the context of Hadoop and MapReduce. Researchers have diligently explored methods to enhance job execution efficiency and minimize resource contention. While some approaches prioritize data locality and real-time resource availability prediction, others adopt a multi-objective and multi-constrained approach to optimize task allocation. However, it is evident that the challenges of heterogeneous clusters, varying resource costs, and accurate progress tracking continue to pose promising research questions in the field. These findings collectively underscore the dynamic nature of Hadoop scheduling and the need for innovative solutions to address the evolving demands of big data processing.

3. Proposed Methodology

We have developed an algorithm to optimize load balancing, as shown in Figure 1. This diagram provides an overview of a load-balancing solution tailored to Hadoop. The purpose of a web scraper, resource report server, and monitoring client is to assist in dynamic load balancing, optimize resource utilization, and enhance the performance of the distributed computing environment. The main goal of our algorithm is to introduce a mechanism that increases the efficiency of the Hadoop system through dynamic load balancing.

3.1. Overview

In our approach, the time required to execute tasks is important. For example, consider a scenario where a resource-intensive task needs to be executed in HDFS and the nodes have different resource availability. In such cases, it is important to assign tasks in such a way that larger tasks are prioritized to nodes with sufficient resources and vice versa. To achieve this, it is necessary to implement a dynamic load-balancing strategy aimed at optimizing performance. In practice, the Load Balancer periodically acquires data pertaining to the Hadoop clusters from the Monitoring System. This dataset encompasses crucial metrics such as CPU utilization, memory consumption, and disk space usage for each individual node within the cluster. Subsequently, the Load Balancer uses these data to identify which nodes are under-utilized and which are struggling with excessive workloads.

3.2. Resource Report Server

The resource Report Server is a main component in our approach that tracks the resources available in a Hadoop cluster. It consists of two submodules: the resource report server and the monitoring client. The resource report server runs on each node in the cluster. It collects information about the node's resources, such as the CPU frequency, the number of cores, the total amount of RAM, and the total amount of storage. The resource report server then sends this information to the monitoring client, which runs on the master node. The monitoring client keeps track of the resources available in the cluster. It uses this information to make decisions about how to allocate resources to jobs. The monitoring client updates the resource tracker table every 10 seconds. The CPU power of a node is defined by Equation (1), and the RAM capability is defined by Equation (2). These equations are used by the monitoring client to calculate the resources available in the cluster.

$$C_{cpu}(i) = f_{cpu}(i) \times N_{cores} \tag{1}$$

This equation is used to calculate the total processing capacity of a given node. To do this, it multiplies the CPU frequency of the cores of that node ($f_{cpu}(i)$) by the total number of CPU cores (N_{cores}). Essentially, this quantifies the node's ability to perform computational tasks based on its CPU configuration.

$$C_{ram}(i) = R_{size}(i) \tag{2}$$

In equation (2) we determine the RAM capacity (C_{ram}) of the same specific node denoted by "i" It directly corresponds to the RAM size ($R_{size}(i)$) of the node. RAM Capacity is critical because it dictates the amount of data that can be cached for processing.

3.3. Web Scrapper

To obtain the number of blocks and the corresponding nodes of data to be processed, we write a web scrapper. This scrapper collects the IPs of the nodes and the blocks they contain from the Hadoop web interface page using regular expressions.

3.4. Required Parameters

This process allows for the customization of parameters and preconditions when initializing the algorithm. Among these parameters, computing power emerges as the most critical factor for each node. It is quantified using the following equation:

$$N_{power}(i) = \alpha\left(\frac{C_{cpu}(i)}{\left(C_{cpu}(i)\right)}\right) \div \left(C_{cpu}(i) + \beta\left(\frac{C_{ram}(i)}{min\,(C_{ram})}\right)\right) \tag{3}$$

Using this equation, we can calculate the computational power for each node, taking into account the capacities of both CPU and RAM. This parameter can then be used in your algorithm to make informed decisions about task allocation and load balancing based on the capabilities of each node.

The node's load rate, denoted as L_{Node}, is determined using Equation (4) as follows:

$$L_{node}(i) = \frac{U_{storage}(i)}{T_{storage}(i)} \times 100 \tag{4}$$

The load rate of a node, denoted as $L_{node}(i)$, is a measure of how much of its storage capacity is being used, expressed as a percentage. The amount of storage used by node "i", $U_{storage}(i)$, is the volume of storage space that is currently occupied. The total storage capacity of node "i", $T_{storage}(i)$, is the entire storage capacity available on the node.

The load rate of the entire cluster, denoted as $L_{cluster}$, is calculated using the equation provided below:

$$L_{cluster} = \frac{\sum_{i=1}^{n} U_{storage}(i)}{\sum_{i=1}^{n} T_{storage}(i)} \quad (5)$$

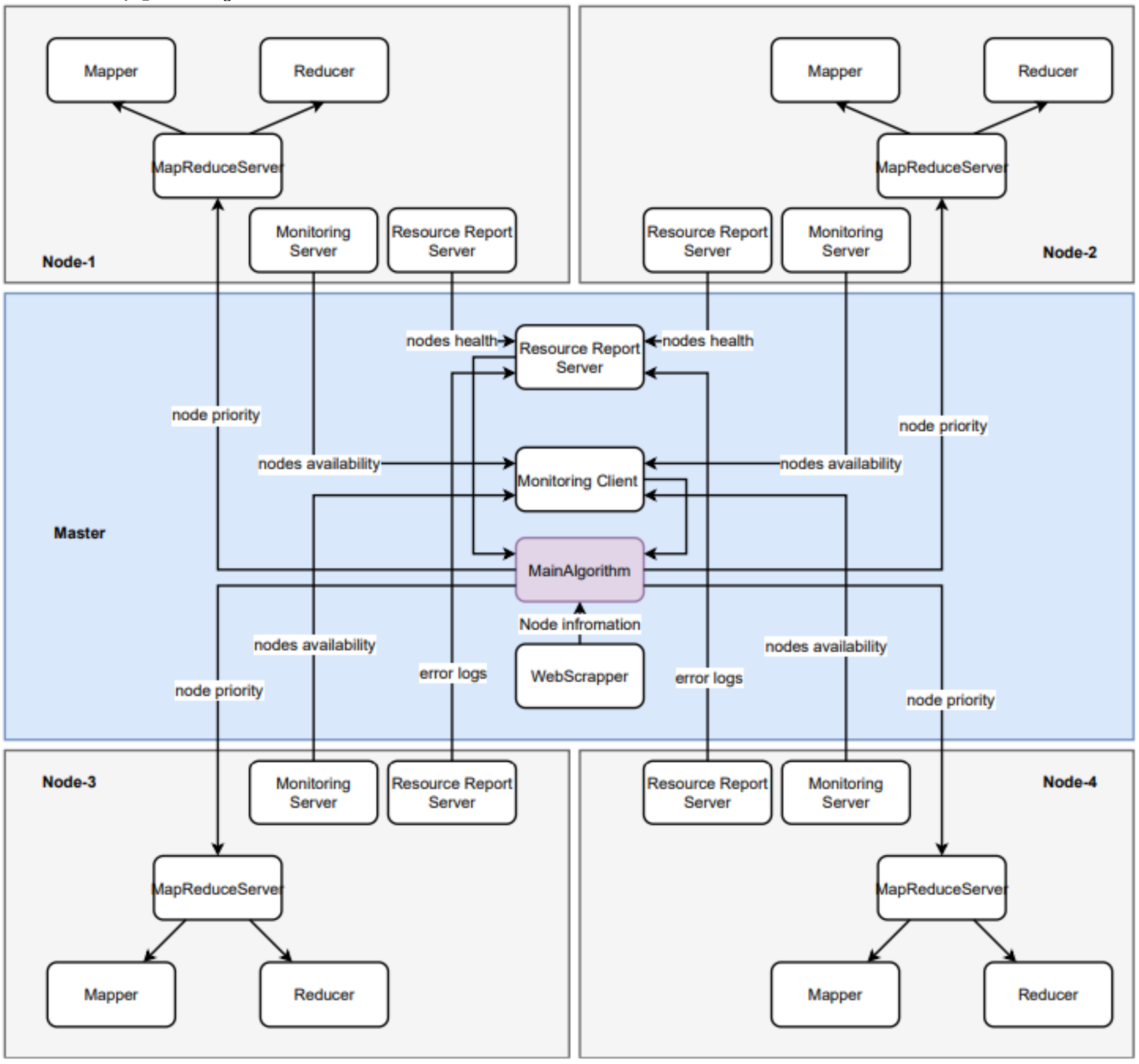

Figure 1. The overview of the proposed approach

The load rate of the entire cluster is represented by $L_{cluster}$. This quantifies the utilization of collective storage at all nodes in the cluster. The amount of storage used by node "i" is denoted by $U_{storage}(i)$, which reflects the volume of storage space that is currently occupied by each node. The total storage capacity of node "i" is indicated by $T_{storage}(i)$, which is the entire storage capacity available on each node.

The maximum load rate of the cluster, denoted as L_{max}, is calculated using the following equation:

$$L_{max} = [\partial + (1 - \partial) \times L_{cluster}] \times 100 \quad (6)$$

The equation combines ∂ and $L_{cluster}$ to calculate the maximum load rate of the cluster. When the cluster is completely occupied (i.e., ∂ = 1), the value of L_{max} is determined only by $L_{cluster}$. On the other hand, when the cluster is unloaded (i.e., ∂ = 0), L_{max} is equal to 80, regardless of the value of $L_{cluster}$.

The hit rate of an individual node, represented as L_{hit}, is calculated using the following equation:

$$L_{hit}(i) = \left(\frac{N_{power}(i) \cdot U_{storage}(i)}{(N_{power} \cdot U_{storage})}\right) \times 100 \quad (7)$$

The equation computes the hit rate for each node by combining computing power and storage utilization. It then scales the result to a percentage by multiplying it by 100. This metric provides insights into how efficiently each node is using its resources within the cluster.

3.5. Dynamic Load Balancer

We have introduced a dynamic load-balancing algorithm tailored for Hadoop. This algorithm dynamically prioritizes job nodes to attain load balancing. After successfully assigning one task, it initiates the allocation of the next task. This iterative process continues until all tasks are allocated.

As shown in Algorithm, it initiates by obtaining monitoring information denoted as$monitoring_{info}$. It then generates the node-priority-list as its output. The algorithm invokes the *getParams()* function with $monitoring_{info}$ as input. This function computes essential parameters, including the load rate of individual nodes ($L_{node}(i)$), the load rate of the entire cluster ($L_{cluster}$), the hit rate of nodes ($L_{hit}(i)$), and the maximum load rate of the cluster (L_{max}). The proposed algorithm algorithm creates three priority tables: *PT1, PT2*, and *PT3* using the function *createPriorityTables()*. These tables store node identifiers along with their corresponding hit rates. Each node is assigned to one of these priority tables based on its node load. If $L_{node}(i)$ exceedsL_{max}, the node is placed in *PT3*. If $L_{node}(i)$ is greater than$L_{cluster}$, the node is assigned to *PT2*. Otherwise, it is assigned to *PT1*. The algorithm first analyzes *PT1*. If it is not empty, it sorts the nodes in descending order based on their hit rates. It then iterates through all nodes in *PT1*. If a node's storage capacity *(Storagenode)* is greater than or equal to the storage required by a task *(Storagetask)*, that node is added to the node-priority-list. In case *PT1* is empty, the algorithm checks *PT2*. If it contains nodes, it sorts them in descending order of hit rate and iterates through the nodes in *PT2*. If a node's storage capacity (*Storagenode*) meets or exceeds the storage requirements of a task (*Storagetask*), it is added to the node-priority-list. This dynamic load-balancing algorithm ensures that tasks are allocated to nodes based on their load rates and storage capacities, achieving efficient load distribution across the cluster.

Algorithm 1: Dynamic Load Balancing Algorithm for Hadoop

```
procedure PRIORITIZENODES
Inputs:
monitoring_info
Output:
node_priority_list
L_node($i$), L_cluster, L_hit($i$), L_max ← getParameters(monitoring_info)
$PT_1, PT_2, PT_3$ ← createPriorityTables()
for each node do
    if $L\_node(i) \geq L\_max$ then
        $PT_3$ += $i$, L_hit($i$)
    end
    else
        if $L\_node(i) > L\_cluster$ then
            $PT_2$ += $i$, L_hit($i$)
        end
        else
            $PT_1$ += $i$, L_hit($i$)
        end
    end
end
if $PT_1$ is not null then
    for each node in $PT_1$ do
        node_priority_list += node
    end
end
else
    if $PT_2$ is not null then
        for each node in $PT_2$ do
            node_priority_list += node
        end
    end
end
return node_priority_list
```

4. Experimental Results

In this section, we have compared the proposed algorithms with existing load balancing algorithms such as FIFO, capacity, fair, and GPA. These algorithms are used in existing Hadoop platforms.

4.1. Dataset

We used the MovieLens 20M benchmark datasets 1 , which comprises 20000263 ratings 318 and 465564 tag applications in 27278 movies. 138493 users created this data set between 319 January 9, 1995 and March

31, 2015. The data set was published on October 17, 2016. All 320 users had rated at least 20 movies. We write extensive resource-hungry tasks that include 321 sorting out the movies according to the ratings, getting the top-rated and worst movies, and 322 recommending movies to the users according to the ratings they had given to the movies.

4.2. Experimental setup

The experiments were conducted on a heterogeneous cluster consisting of one master node and four slave nodes, each differing in hardware resources. This configuration was deliberately chosen to facilitate a comprehensive examination of time efficiency. Detailed information on the hardware resources for each node can be found in Table 1.

Table 1. Hardware Resources of Each Node.

Sr No	Cores	CPU Freq	RAM	HDD	Responsibility	IP Address
1	3	2.60 GHz	5 GB	30 GB	Master	192.168.56.102
2	2	2.60 GHz	4 GB	20 GB	Slave_1	192.168.56.103
3	1	2.60 GHz	3 GB	12 GB	Slave_2	192.168.56.104
4	1	2.60 GHz	2 GB	9 GB	Slave_3	192.168.56.105
5	1	2.60 GHz	1 GB	8 GB	Slave_4	192.168.56.106

Our implementation was based on a Linux-based system, and we configured Apache Hadoop version 3.3.1 on both the master and slave nodes to ensure seamless integration with Hadoop. To enable efficient communication between the master and slave nodes, we disabled the firewall. We conducted four replications and recorded the outcomes of our results.

For the remote distribution of jobs from the master node to the slaves, we leveraged the well-known open-source Python library, Pyro4. Within this setup, the Pyro4 name server was configured on the master node, and relevant jobs were registered. MapReduce tasks were submitted to the master node, which effectively managed the resources within the cluster.

4.3. Evaluation Results

In this experiment, we compared our approach with state-of-the-art approaches as shown in Figure 2. The X-axis indicates the number of films processed simultaneously, while the Y-axis indicates the job completion time in minutes. Examination of the results shows that the proposed algorithm is more efficient when running larger jobs. The proposed dynamic load balancing algorithm is compared with the default load balancing algorithms, the FIFO scheduler, the capacity scheduler, the fair scheduler and the previously proposed greedy priority algorithm [27]. As shown in Figure 2 the proposed algorithm has the shortest completion time of the job and

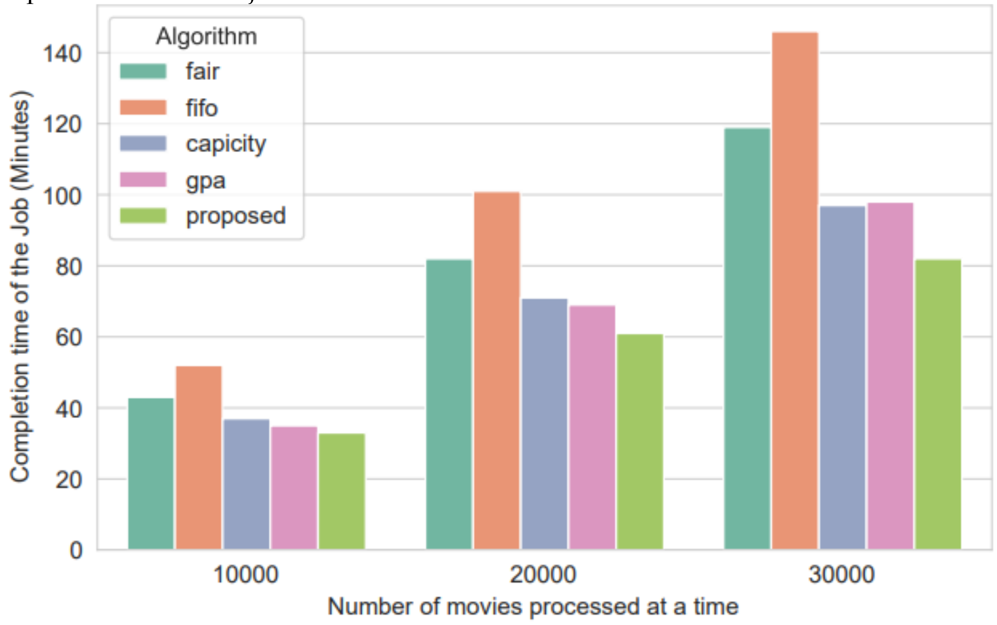

Figure 2. Comparison of the completion time of a job for different load balancing algorithms

The x-axis shows the number of movies processed at a time, and the y-axis shows the completion time of the job in minutes. The bar represents the different load balancing algorithms with less resource-hungry jobs w.r.t. time shows that the proposed algorithm is the best strategy for load balancing. The fair algorithm has the second shortest completion time, followed by the FIFO algorithm, the capacity algorithm, and the GPA algorithm. The FIFO algorithm is the simplest load balancing algorithm. It simply processes the jobs in the order in which they arrive. The capacity algorithm assigns jobs to servers based on server capacity. The GPA algorithm assigns the jobs to the servers based on the GPA of the servers. The proposed algorithm is a more complex algorithm that considers the number of movies processed simultaneously. This allows it to find a better solution for load balancing, resulting in shorter completion times. We found that the proposed dynamic load balancing algorithm can reduce the execution time of jobs and effectively balance the load, resulting in higher cluster efficiency. The load on each cluster is monitored, and the job execution time is calculated.

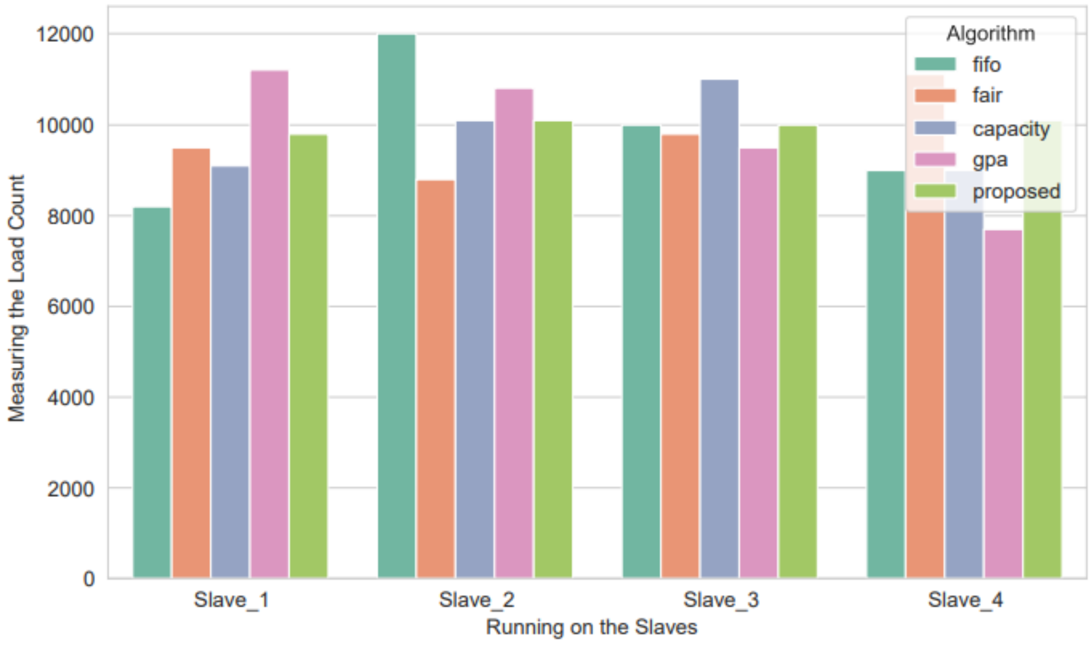

Figure 3. Showing the load comparison on different nodes and x-axis represents the nodes, while the y-axis represents the load count

We conducted another experiment to compare our approach with existing methods. In this experiment, we assessed the distribution of load among the cluster nodes by analyzing monitoring module logs and calculating node loads at various intervals. The results are presented graphically in Figure 3, with the X-axis representing nodes and the Y-axis representing cluster loads. The findings clearly demonstrate that our algorithm outperforms FIFO, Capacity, Fair, and GPA in terms of load balancing. Figure 3 is a bar plot that visually depicts load counts for each load balancing algorithm. It's noteworthy that our proposed algorithm consistently shows lower load counts across all slave nodes, highlighting its scalability as a load balancing strategy based on load ratios.

5. Discussion

The experimental results presented in this research provide valuable insights into the efficiency and effectiveness of the proposed dynamic load-balancing algorithm for Hadoop clusters. The evaluation aimed to compare the proposed algorithm with existing algorithms and assess its impact on job completion times and load distribution. One of the key findings is that the proposed dynamic load balancing algorithm outperforms the default load balancing algorithms (FIFO, capacity, fair) and the previously proposed GPA algorithm in terms of job completion time. This is particularly significant in large-scale data processing scenarios, where efficient job completion directly impacts the overall throughput and productivity of the system. For instance, reduced job completion times could enable organizations to generate reports and insights more quickly or to train text clustering or machine learning models more efficiently.

The experiment also highlights the efficiency of the algorithm in processing larger jobs, emphasizing its scalability and potential to handle substantial workloads. This is crucial in real-world applications where data processing tasks can vary significantly in size and complexity. For example, the algorithm could be employed to process large datasets for scientific research or to power real-time data analytics applications. The ability of the algorithm to consider the number of movies processed simultaneously showcases its adaptability and responsiveness to varying workloads. Furthermore, the experiment analyzing the load distribution among the cluster nodes demonstrates that the proposed algorithm effectively balances the load across the cluster, as evidenced by the lower load counts on all slave nodes.

This finding underscores the algorithm's capacity to efficiently utilize available resources, ensuring a more balanced and optimized system. For instance, this could assist organizations in reducing costs and improving the performance of their Hadoop clusters. The implications of these findings are significant. Efficient load balancing directly translates to improved performance and resource utilization within Hadoop clusters. Shorter job completion times mean quicker data processing, enabling organizations to derive insights and make decisions faster. Moreover, the scalability and adaptability of the proposed algorithm render it a promising solution for diverse Hadoop cluster configurations and varying workloads. The results and findings of this research provide compelling evidence for the effectiveness and efficiency of the proposed dynamic load-balancing algorithm in the context of Hadoop clusters.

The algorithm has the potential to significantly enhance Hadoop performance by optimizing resource allocation, improving load balancing, and ultimately reducing costs and resource consumption. Future research can build upon these findings, exploring further refinements and applications of the algorithm to continue advancing the field of big data processing. For instance, future research could focus on refining the algorithm to improve its performance in specific scenarios, such as when handling large numbers of concurrent jobs or when dealing with heterogeneous clusters. Additionally, future research could investigate new applications for the algorithm in different domains, such as machine learning and real-time data analytics.

6. Conclusions

In this manuscript, we present a dynamic load balancing algorithm engineered to enhance the efficiency of Hadoop clusters. Our empirical assessments underscore that this innovative algorithm substantially diminishes job processing durations while sustaining stability, particularly for sizeable tasks. The proposed model integrates a monitoring module that persistently tracks node performance metrics, supplemented by a distinctive method for task-node allocation adjustments, thereby guaranteeing load equilibrium within the cluster.

When juxtaposed with existing strategies, our algorithm not only realizes quicker completion times relative to the greedy algorithm but also exhibits double the efficiency of FIFO. Furthermore, in terms of scalability, our methodology surpasses the Fair, Greedy, and Capacity schedulers, significantly alleviating the burden on master nodes. This technique proficiently equilibrates cluster loads, expedites task completion, and mitigates costs and resource consumption, as evidenced by our experimental results. The advantages of our method become increasingly evident with the augmentation of cluster size and complexity in heterogeneous scenarios.

Future inquiries in this domain should emphasize investigating load balancing in varied clusters, particularly within heterogeneous environments. Examining the influence and optimization of assorted rack and replication mechanisms is crucial for further augmenting cluster performance and efficiency.

References

1. Singh, T.; Gupta, S.; Kumar, M.; et al. Adaptive load balancing in cluster computing environment. The Journal of Supercomputing 2023, pp. 1–29.
2. Dean, J.; Ghemawat, S. MapReduce: simplified data processing on large clusters. Communications of the ACM 2008, 51, 107–113
3. Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. The hadoop distributed file system. In 436 Proceedings of the 2010 IEEE 26th symposium on mass storage systems and technologies 437 (MSST). Ieee, 2010, pp. 1–10.
4. White, T. Hadoop: The definitive guide; " O'Reilly Media, Inc.", 2012.).
5. Zaharia, M. Job scheduling with the fair and capacity schedulers. Hadoop Summit 2009, 9, 592.
6. Isard, M.; Prabhakaran, V.; Currey, J.; Wieder, U.; Talwar, K.; Goldberg, A. Quincy: fair scheduling for distributed computing clusters. In Proceedings of the Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, 2009, pp. 261–276.
7. Kumar, Y.; Kaul, S.; Hu, Y.C. Machine learning for energy-resource allocation, workflow scheduling and live migration in cloud computing: State-of-the-art survey. Sustainable Computing: Informatics and Systems 2022, 36, 100780.
8. Zhang, Z.; Zhang, X. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In Proceedings of the 2010 The 2nd international conference on industrial mechatronics and automation. IEEE, 2010, Vol. 2, pp. 240–243.
9. Radojevi´c, B.; Žagar, M. Analysis of issues with load balancing algorithms in hosted (cloud) environments. In Proceedings of the 2011 Proceedings of the 34th international convention MIPRO. IEEE, 2011, pp. 416–420.
10. Wu, T.Y.; Lee, W.T.; Lin, Y.S.; Lin, Y.S.; Chan, H.L.; Huang, J.S. Dynamic load balancing mechanism based on cloud storage. In Proceedings of the 2012 Computing, Communications and Applications Conference. IEEE, 2012, pp. 102–106.
11. Asan Baker Kanbar, K.F. Modern load balancing techniques and their effects on cloud computing. Journal of Hunan University Natural Sciences 2022, 49.
12. Ni, J.; Huang, Y.; Luan, Z.; Zhang, J.; Qian, D. Virtual machine mapping policy based on load balancing in private cloud environment. In Proceedings of the 2011 International Conference on Cloud and Service Computing. IEEE, 2011, pp. 292–295.
13. Wang, S.C.; Yan, K.Q.; Liao, W.P.; Wang, S.S. Towards a load balancing in a three-level cloud computing network. In Proceedings of the 2010 3rd international conference on computer science and information technology. IEEE, 2010, Vol. 1, pp. 108–113.
14. Mahalle, H.S.; Kaveri, P.R.; Chavan, V. Load balancing on cloud data centres. International Journal of advanced research in computer science and software engineering 2013, 3.
15. Soni, G.; Kalra, M. A novel approach for load balancing in cloud data center. In Proceedings of the 2014 IEEE international advance computing conference (IACC). IEEE, 2014, pp. 807–812.
16. Adhikari, J.; Patil, S. Double threshold energy aware load balancing in cloud computing. In 468 Proceedings of the 2013 fourth international conference on computing, communications and 469 networking technologies (ICCCNT). IEEE, 2013, pp. 1–6.
17. Gautam, J.V.; Prajapati, H.B.; Dabhi, V.K.; Chaudhary, S. A survey on job scheduling algorithms 471 in big data processing. In Proceedings of the 2015 IEEE International Conference on Electrical, 472 Computer and Communication Technologies (ICECCT). IEEE, 2015, pp. 1–11.
18. Hammoud, M.; Sakr, M.F. Locality-aware reduce task scheduling for MapReduce. In Proceed- 474 ings of the 2011 IEEE Third International Conference on Cloud Computing Technology and 475 Science. IEEE, 2011, pp. 570–576.
19. Kumar, K.A.; Konishetty, V.K.; Voruganti, K.; Rao, G.P. CASH: context aware scheduler for 477 Hadoop. In Proceedings of the Proceedings of the international conference on advances in 478 computing, communications and informatics, 2012, pp. 52–61.
20. Zhang, X.; Zhong, Z.; Feng, S.; Tu, B.; Fan, J. Improving data locality of mapreduce by scheduling 480 in homogeneous computing environments. In Proceedings of the 2011 IEEE Ninth International 481 Symposium on Parallel and Distributed Processing with Applications. IEEE, 2011, pp. 120–126.
21. Hou, X. Dynamic Workload Balancing and Scheduling in Hadoop MapReduce with Software 483 Defined Networking. PhD thesis, Oklahoma State University, 2017.
22. Yong, M.; Garegrat, N.; Mohan, S. Towards a resource aware scheduler in hadoop. In 485 Proceedings of the Proc. ICWS, 2009, pp. 102–109.

23. Nanduri, R.; Maheshwari, N.; Reddyraja, A.; Varma, V. Job aware scheduling algorithm for 487 mapreduce framework. In Proceedings of the 2011 IEEE Third International Conference on 488 Cloud Computing Technology and Science. IEEE, 2011, pp. 724–729.
24. Mao, H.; Hu, S.; Zhang, Z.; Xiao, L.; Ruan, L. A load-driven task scheduler with adaptive DSC 490 for MapReduce. In Proceedings of the 2011 IEEE/ACM International Conference on Green 491 Computing and Communications. IEEE, 2011, pp. 28–33.
25. Li, Y.; Zhang, H.; Kim, K.H. A power-aware scheduling of mapreduce applications in the cloud. 493 In Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic 494 and Secure Computing. IEEE, 2011, pp. 613–620.
26. Kc, K.; Anyanwu, K. Scheduling hadoop jobs to meet deadlines. In Proceedings of the 2010 496 IEEE Second International Conference on Cloud Computing Technology and Science. IEEE, 497 2010, pp. 388–392.
27. Chen, R.; Zeng, W.H.; Fan, K.J. Research on Hadoop Greedy Scheduler Based on the Fair. In 499 Proceedings of the Applied Mechanics and Materials. Trans Tech Publ, 2012, Vol. 145, pp. 500 460–464.