Journal of Data Acquisition and Processing
ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.777643

EFFICIENT RESOURCE MANAGING AND JOB SCHEDULING IN A HETEROGENEOUS KUBERNETES CLUSTER FOR BIG DATA

Jayanthi M[*1], Dr. K. Ram Mohan Rao[2]
Department of Computer Science and Informatics, Mahatma Gandhi University, Nalgonda, India
Department of Information Technology, Vasavi College of Engineering, Hyderabad, India
Email: jayanthimgu343@gmail.com, krmrao@staff.vce.ac.in

Abstract: Cloud computing is an on-demand model of computing that utilizes virtualization expertise to offer cloud resources such as CPU, memory, storage, and network to customers in the usage of virtual machines. As a result, most big data analytics in many modern enterprise applications are run from the cloud. Since resources in these private clouds are limited, getting the most out of resource applications and providing guaranteed service to users is the ultimate goal by efficiently scheduling tasks and resources. However, existing schedulers in big data processing systems do not consider both application performance and resource utilization when performing allocations. Therefore, it is difficult to design workflows to accomplish low turnaround time and high resource consumption in big data systems. In this paper, we propose a resource management system for efficient job scheduling, called RMS, which dynamically schedules big data jobs in Kubernetes cluster nodes for Spark applications, and autonomously adjusts scheduling policies in heterogeneous node clusters to enhance application execution and resource consumption. The RMS mechanism will ensure that there is sufficient guidance and resources available in its planning objectives and a satisfactory level of resource utilization. The experimental analysis of different RMS and performance preferences using different methods depends on the predicted completion time and the benchmark statistical result of different big data performance indicators traces. The results show that RMS decreases the cost and scheduling overhead and improves job execution performance.

Keywords: Cloud computing, Big Data, Job Scheduling, Resource Management, Kubernetes, Spark.

1. Introduction

Cloud computing is an innovative, cost-effective form of underwriting that is rapidly becoming a transformational technology for many organizations due to its flexibility and use of virtual resources as online services [1]. Here, computing power and memory are provided to users in the form of virtual machines to provide virtual servers with predefined configurations [2-3]. With the rise of big data analytics and artificial intelligence, big data analysis (BDA) has played an important role in today's business [4]. Therefore, minimizing the underutilization of underlying resources is critical to ensuring effective efficiency through efficient resource management and scheduling. However, a major scheduling challenge in a

cloud environment is efficiently allocating resources to improve the performance of the overall cloud computing environment.

Currently, most cloud service providers rely on simple resource allocation, such as fast and optimal deployment [5]. Advanced scheduling methodology is a well-studied phenomenon in network environments, but due to the dynamic nature of incoming requests, fast best effort is preferred in public clouds [6]. The main reason cloud applications lack predictable or predetermined usage patterns is their dynamic nature. Therefore, modern scheduling techniques continue to be limited to the grid and are not considered suitable for public clouds. However, this scenario is somewhat different for enterprise clouds, where usage patterns are at least somewhat predictable. This can be attributed to the fact that the private cloud is owned by an organization and managed and used primarily by employees and other stakeholders [6-7]. We, therefore, believe that in a private cloud with predictable availability, scheduling with different rules will improve resource utilization and ensure services.

The job scheduler is also an important part of BDA, where multiple tasks are performed by multiple users and applications compete for resources in a collective environment. Many BDAs are often limited as the volume of data increases and the need for analytics increases. Consequently, effective resource management becomes an important function task of the cluster scheduler [8]. Incoming job requests from various applications require heterogeneous resources. The usual resources for a virtual machine are the number of cores, CPU, memory, and bandwidth required to operate. Therefore, before scheduling can be done effectively, incoming tasks must be classified. Many research works highlight the need to explore avenues for dynamic resource scheduling methods in the cloud. In this case, the resource utilization of nodes can be maximized and better service guarantees can be provided for incoming requests [9]. In this paper, we propose a resource management system (RMS) for efficient job scheduling in heterogeneous node clusters according to resource heterogeneity requirements. Incoming requests are submitted by a Spark application in a GC platform virtual machine running a Kubernetes cluster node [10]. Each node in the cluster is configured with heterogeneous resources to assign different jobs. In this case, proper resource allocation is very important to achieve cost efficiency. RMS considers job requirements, assigns jobs to appropriate nodes first, and adjusts scheduling strategies in heterogeneous node clusters to increase application performance and resource consumption. It incorporates work demand and resource availability in its scheduling decisions while maintaining a satisfactory level of resource utilization. The main contribution of our proposed work is as follows:

(i) Identify the resource utilization of cluster nodes through resource monitoring,
(ii) An Efficient Job Scheduling Algorithm Based on Classification of Job Resource Requirements of Heterogeneous Cluster Nodes.
(iii) Increase the cost-effectiveness of your RMS through the optimal distribution of work.

An efficient scheduling program will achieve higher results but use more resources. The number of cores and the amount of memory allocated to the application task are considered as per the node capacity for the scheduling algorithm. It is important to allocate products to specific projects as quickly as possible, based on customer and service provider satisfaction.
The rest of the paper is organized as follows. It gives a brief introduction to the related works in Section-2. Section-3 it presents our proposed RMS and scheduling algorithm. Section-4

provides the experimental setup and results analysis of our work using a big data benchmark dataset and Section-5 discusses the conclusion of our proposal.

II. Related Works

Many activities have been investigated in cloud computing to reduce the request rejection rate, maximize revenue, and improve resource utilization [13, 23, 27, 34, 35]. To achieve these objectives, scheduling algorithms often focus on group requests or potential resources using heuristic methods. Much research has focused on scheduling in cloud environments, where advanced resource booking techniques are well considered. The scheduling function of BDA has received much attention from various domains and academic studies. In terms of scheduling objectives, previous studies can be classified into two categories: performance-based and fairness-based scheduling.

A. Scheduling based on Performance

The scheduler's performance is determined by the number of jobs completed within a specified deadline compared to the number of job requests during a specified time interval. Many methods have been proposed in the past to predict the performance of schedulers based on generic and procedural models to achieve an accurate assessment of the accomplishment time of big data analytics jobs.

Dinn et al. [11] presented an improvised solution for multi-object scheduling known as “Harris hawks optimizer (HHO)” for solving the issue of allocation of multi-objects. Khan et al. [12] suggest an efficient scheduling approach to reduce the job waiting time by utilizing a hybrid optimization algorithm. Meyer et al. [13] with support of a defined taxonomy, illustrate that resource scheduling based on workload variation presents a machine, learning-based classification model. Its objective is to assign jobs dynamically with interfering job resource requirements. Chen et al. [14] utilize a heterogeneous environment for efficient scheduling for data centers by considering the VM cores and energy utilizations based on variations in the resource occurrence needs.

The growing demand for cloud resources of data centers increases the challenges to meet the demand performance in real time. It very much needs new solutions to handle such a huge inflow of requests. Cheng et al. [15] provide a learning method for real-time job scheduling using Deep Reinforcement Learning to mitigate the server workload and smoothly execute jobs. Fan et al. [16] suggest a methodology for diverse workloads utilizing an intelligent scheduling system for job scheduling based on hardware resources for different kinds of machines. Zheng et al. [17] developed a method for the online user to avail SaaS and IaaS optimally to manage their required jobs efficiently or execute the jobs in parallel to resolve the requirements. Shao et al. [18] suggest a greedy method to reduce the time of execution and resource consumption in terms of power for allocating jobs in a fine-grained manner. So, to have a modest enhancement in performance in job execution a continuous observation of the demand with the least resource and cost is essential [19].

To meet the job execution scheduling and estimation of its performance different approaches are explored [23], and also it is very much important to know the deadline before scheduling the jobs. Hou et al. [20] present a method to reduce the average time of job execution to meet the deadline of jobs. It provides a deadline-aware scheduling method by

monitoring the job resource requirement and estimation of job completion time. Wang et al. [21] also suggest a method with the understanding of the job workflow model and deadline restrictions to meet cost-effectiveness. Yao et al. [22] exploit the resource utilization for different types of jobs and their dependency to enhance the availability of the resource for jobs and meet the deadline of jobs.

B.. Scheduling based on Fairness

Fairness is another important element of a scheduling framework for modifying the behavior of long-waiting tasks by dynamically balancing the performance of diverse jobs for rational and fast accomplishment. An adjustment of the appropriate scheduling policies based on the workloads to run for better and fairer performance is essential. A generic meta-scheduler based on Hadoop YARN's [24] is implemented to achieve the fairness trade-off of the schedulers.

Wang et al. [25] use multiple metrics to effectively balance efficiency and fairness by reducing the runtime cost of MapReduce jobs. Zaharia et al. [26] offer a delayed scheduling policy in Hadoop to increase the data space and improve the performance of the Fair Scheduler. Work to demonstrate minimum-to-maximum fairness over multiple resources using a Hadoop YARN clustering for resource fairness is given as DRF [27]. Tang et al. [28] to enhance schedule fairness present a slot allocation process in the Hadoop framework to dynamically allot the workload.

Selvarani et al. [29] present an improved cost-based scheduling approach for efficient resource allocation by grouping projects according to the most profitable projects. The purpose of this calculation is based on the profitability of the service provider rather than the satisfaction of the users. Li et al. [30] proposed a feedback preemptive task scheduling approach for scheduling a project with the shortest mean run time. However, the proposed algorithm can lead to starvation because longer-running tasks are held in a high-latency queue. Huang et al. [31] maximized the client-specified utility concerning max-min fairness by calculating an approximate total workload based on the run-time distribution of tasks and performing resource allocation accordingly.

Yang et al. [32] suggested a heuristics approach for task assignment that assigns all tasks in random order with the minimum completion time of virtualization resources. This algorithm only utilizes the allotted time of the VM, not its resource. In this way, VMs on different hosts are provisioned and the resources used are reduced. Ghanbari et al. [33] studied the statistical methods in cloud computing to analyze the job priority to do accurate job scheduling. Each activity requires predetermined resource priority. An association matrix was calculated for each task based on resource accessibility to decide the priority. A priority or weight vector is computed for each associated matrix, and its resources are allocated relying on these priority vectors, but it also shows a limitation in deadline violations in the job completion.

Sharkh et al. [34] proposed a framework for making advanced requests blocking over software-defined networks (SDN) for distribution job allocation purposes. A greedy algorithm which divide the start time in advance, so that it can minimize the job allocation delay. The algorithm only considered the pattern of resources to public cloud demand and limited the schedule of the demand to the advanced reservation type.

An advance scheduling methodology in support of predefined allocation constraints for allocating jobs as per their immediate need, best fit meets deadline importance, or advance resource booking is presented by Nathani et al. [35]. In this allocation when a request arrives, the scheduler will attempt to process the request at one or more locations. If it was unable to find any then it tries to reschedule the request by changing the existing schedule and employing a swapping and backfilling procedure. During an exchange, two consecutive jobs are exchanged only if the first job requires less resource than the second and no timeouts have elapsed since the swapping. The backfill procedure is used to reserve best fit and deadline jobs by re-booking several idle slots. This procedure is used to boost idle resources during new job demand periods. The exchange of best-fit requests with a deadline in this way is starved because there is no indication of the number of times this swapping will accomplish the task.

Inspired by these findings, we outline that the different guidelines and procedures to improve starvation eradication for the best-fit job allocation are needed through continuous resource monitoring and scheduling. In addition, resource utilization is improved by efficiently identifying and scheduling idle resources using the proposed scheduling algorithm.

III. Proposed System

The proposed resource management system (RMS) focuses on the job scheduling problem through node resource monitoring. We considered heterogeneous nodes in the cluster for running BDA jobs. Other scheduling frameworks such as YARN and Mesos are in use, but the framework determines possible resource provisioning schemes and provides resources based on weights, quotas, or roles. Here, the scheduler is accountable for refusing or accepting the resources presented allowing the scheduling policy. RMS minimizes denials and improves service by predetermining node resources for tasks to be allocated. RMS implements resource management and job scheduling modules for efficient job execution.

A. Resource Management

A Kubernetes cluster has standard computing resources such as CPU, memory, storage, and network to accommodate the different BDA job requests for execution. For example, jobs executing on the YARN scheduler must proclaim the CPU cores and memory required when the request is submitted. It is considered as the resource prerequisite of a job denoted as $r = \{c, m\}$, where c is CPU core counts and m is the memory, and the total resource of a node is denoted as $R = \{C, M\}$.

For resource monitoring of cluster node resources, it collects job information running on the cluster nodes and prepares a resource availability vector (RAV) from the information obtained from the nodes. The RAV contains a set of records having information of {*Node-Id; Number of Free Core; Free Memory Size*}.

To build RAV, we compute the Job Availability Time (JAT) for a job in comparison to running job core and the number of free cores for a period and modify the RAV every time a new job is allocated to a node or as soon as a job finishes and free resources. The JAT can be calculated using Eq. (1).

$$JAT(rC_t, rM_t, ET_t) = ((C_t - rc_t), (M_t - rm_t), (T(\mathrm{N}_t))) \quad (1)$$

where, rC_t, rM_t, and ET_t is denotes the current Core resource, Memory resource, and estimated time for the availability of a node N_t at a time interval T. The total count of cores and total memory of a node is denoted by C_t and M_t, whereas the current usage of core and memory is denoted by rc_t and rm_t. The value of rc_t and rm_t is computed based on the summation of the number of cores underutilize for running the jobs and the quantity of memory under usage.
In case no free resources are available then it will compute the probable job completion time and compute JAT with an additional waiting delay constant as ω using Eq. (2).

$$JAT(rC_t, rM_t, ET_t) = (C(job_i)_t, M(job_i)_t, (FT(job_i)_t + \omega)) \quad (2)$$

where $C(job_i)_t$ and $M(job_i)_t$ are denoted by the number of cores and memory utilized and $FT(job_i)_t$ denotes the completion time by a job job_i.
Using Eq. (1) and (2) RAVs are created periodically and modified every time variations occur. RMS must refer to and use information about resources available in RAV to make job assignment decisions. The job scheduler component of RMS learns an appropriate node for a receiving request and allocates the request to that node for running. The scheduler takes the RAV information and applies Eq. (3) to determine the node to allocate the job.

$$JAT(N_t) = (free(rC_t), free(rM_t), lowest(ET(job_i)_t)) \quad (3)$$

The JAT(N_t) gives the node the number of cores free and the amount of memory free for the lowest expected time, so it can schedule the subsequent incoming job request along with the resource requirements to meet the execution.
Here, RMS provides the necessary resources from available resources on the node to satisfy the task execution, but usually, a cluster node may have multiple tasks running at specific time intervals under preemption conditions that can cause tasks to wait a long time. Therefore, it is important to efficiently schedule and allocate jobs in the most appropriate way to minimize waiting delays for jobs.

B. Job Scheduling

In general, incoming work requests consist of application logic to run when resources are needed. It is queued until reserved for a node. Job scheduling in a homogeneous environment can be FIFO efficient, but in a heterogeneous environment, it is essential to adapt to resource availability to be cost-effective.

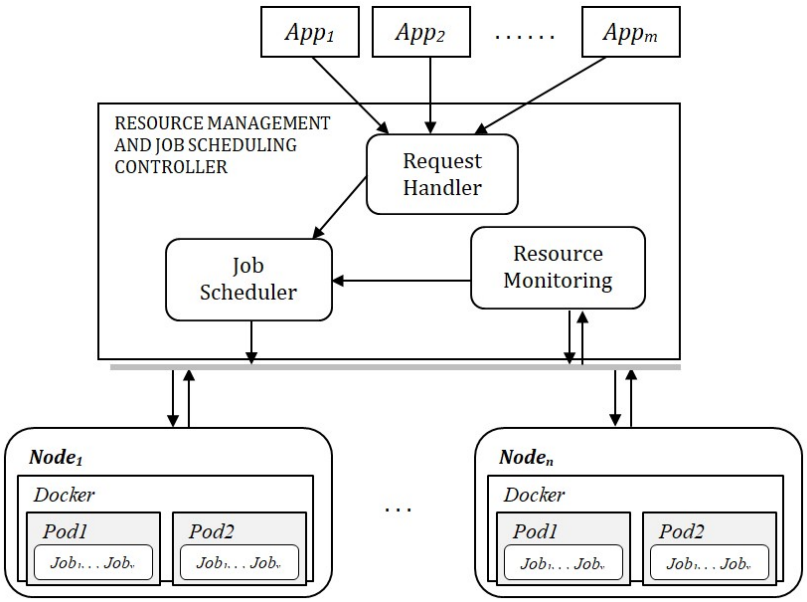

Figure.1: Resource Monitoring and job scheduling system

The proposed resource management and job scheduling system is shown in Figure 1. It works with Request Handler (RH), Resource Monitoring (RM), and Job Scheduling (JS) modules. Work requests coming in from different BDAs are forwarded to JS for processing by RH and assigned to the most appropriate node available. Nodes in a cluster are organized in three sizes based on CPU cores: medium, high, and X-high. Each node consists of x cores that are assigned tasks according to their requirements.

Upon receiving a job request, RH acquires the job information of the required resources in terms of CPU cores and memory and sends it to JS. Here, JS uses a resource-based optimal scheduling (RB-BFS) algorithm to predict optimal node capacity. Scheduling that supports node information provided by RM. A node with multiple cores utilizes the cluster's resources in units of cores, allowing a node to be utilized for multiple tasks concurrently. Therefore, assigning multiple tasks to the same node can lead to the empty packing problem [36], which can be solved by a distributed resource-sharing model within nodes.

Let's assume the cluster has m configured nodes. Each node N_t can allocate up to n jobs at any time, and each task ji requested from the system may require a service that requires x cores and y memory to execute. Each node capacity is featured by balance capacity, and the first to find the availability of cores and memory for a task is allocated for execution. Subsequently, the interpretation of this is NP-complete, so we use a greedy set heuristic method to obtain the optimal result. We used the RB-BFS algorithm with RAV information obtained through resource monitoring to optimize resource utilization and increase the cost efficiency presented in Algorithm-1.

Algorithm-1: RB-BFS algorithm

Input: JR → Set of incoming job requests.

RAL → Resource availability list of Nodes.

Output: Allocation of Job to a Node.

For every arriving job $ji \in JR \; \forall \; i = 1$ to J

```
{
    //-- the number of CPU cores and memory
    Qi = C(ji) && M(ji);

    //-- Find the best node to allocate
    //-- Evaluate RM data (RAL)
    AssignFlag = False;
    For each node RAL vk ∈ RAL ∀ k = 1 to V
    {
        Wk = C(vk) && M(vk);
        If ( Qi ≤ Wk ) {
            Allocate the Job to node Nk ;
            AssignFlag = True;
            Break;
        }
    }
    If(AssignFlag == False) {
        Place back in the JR queue.
    }
}
```

The algorithm assigns tasks to nodes when the necessary capacity is met by the node's free resource capacity. A node might have several job requests in the queue which need to be scheduled for a period, so it is important to calculate the best fit and best satisfy resource availability for efficiently assigning the job.

IV. Experiment Evaluation

A. Setup

We executed our method and perform extensive experimentation in various test settings. We use Kubernetes on Spark 3.2.1 and perform experiments on cluster nodes configured as shown in Table-1. For evaluation, we set up a default scheduler and comparator for benchmark application data [37]. Evaluation measures are used to measure cost, performance, and overhead. The VM is configured on the GCP platform with 20 GB of storage for $0.24 per hour.

Table 1: Cluster Node Resource Configuration

Resource Instance	**CPU Cores**	**Memory (Gb)**	**Quantity**
Medium	6	16	2

High	10	32	2
X-High	16	48	2

We evaluate the enhancement of the proposed JSM using benchmark data information provided in BigDataBench [37] to create Spark workloads. Further precisely, we select 4 dissimilar workloads known as WordCount (WC), Sort (ST), PageRank (PR), and Mixed (MX) as given by Islam et al. [38]. These workloads vary with the scales of data and resource requirements in the form of Cores and Memory as given in Table 2.

Table 2: Job Type workload and resources required

Job Type	**Workload Scale**	**Resource for Low**	**Resource for High**
WordCount (WC)	Low, High	⟨*4C, 8Gb*⟩	⟨*10C, 16Gb*⟩
Sort (ST)	Low, High	⟨*4C, 8Gb*⟩	⟨*10C, 16Gb*⟩
PageRank (PR)	Low, High	⟨*4C, 8Gb*⟩	⟨*10C, 16Gb*⟩
Mixed (MX)	Low, High	⟨*4C, 8Gb*⟩	⟨*10C, 16Gb*⟩

These actions are extracted from Facebook and Hadoop response traces for these various actions. Incoming jobs are dependent on high and low loads for various periods selecting the resource demands listed in Table 2. There are over 100 job requests during high load and nearly 50 job requests during low load.

B. Baseline Schedulers

The challenge with cluster scheduling methods in Spark jobs is that they don't account for facilitator-level job assignments. These methods primarily focus on selecting the resources or multiple nodes required for every task during scheduling decisions. However, the proposed JSM runs at the pod selectivity level by incorporating resource utilization estimation to efficiently schedule task assignments. Compare the following schedulers to see improvements in the proposal.

- FIFO: Apache Spark's default FIFO scheduler is installed on Apache Mesos. Here, jobs are scheduled based on a first-come first-service. Instead of using the scheduler's merge preference; it distributes executors in a round-robin manner. Most existing scheduling algorithms use this preferred method of placing jobs and choose this scheduler as one of their baselines as it is also a common choice for users using Spark jobs.
- BFD [38]: This proposal is a greedy procedure inherited from the “best-fit Decreasing (BFD)” heuristic to optimize the cost of Apache Spark clusters deployed using Apache Mesos as a cluster scheduler for the applications.

- Morpheus [39]: In this strategy, low-cost packaging is utilized for actuator assignment. Based on the present load in a cluster it employs the strategy to detect the job's resource requirements (such as memory or CPU cores). Later, the jobs are sorted in ascending order as per the limited resource requirements. As a result, the cluster's resource is well-proportioned during the scheduling and allows for running maximum jobs.

For the optimal scheduling in the proposed RMS, we configured the minimum and maximum range of CPU consumption to [0.5 to 0.9] and memory utilization to [0.75 to 1], respectively.

C. Results

A. Cost Efficiency

This evaluation shows that the proposed scheduling algorithm can be applied to diverse kinds of applications while decreasing the cost of employing big data clusters. It stores the RAV of several pods used by the node to compute the sum of the cost acquired by the scheduler. Here, over period *T*, we summed up the number of pods under running for the various jobs in a node. The number of pods utilized in a given period in the node is directly proportional to the running cost. So, the total cost we computed using the given Eq. (4).

$$Total\_Cost = \sum_{t \in T} NumberofPods(t) \qquad (4)$$

It calculates the cost of each scheduling algorithm under low and high workloads to evaluate cost-effectiveness with different resource demands as given in Table-2.

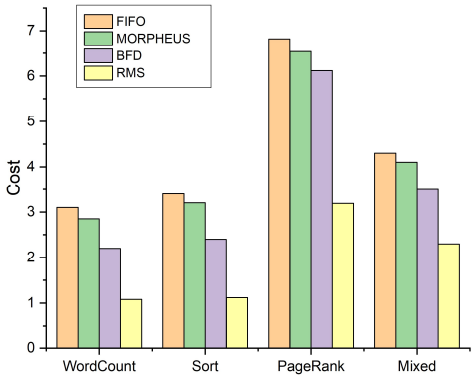

Figure. 2 Cost Efficiency at Low Workload

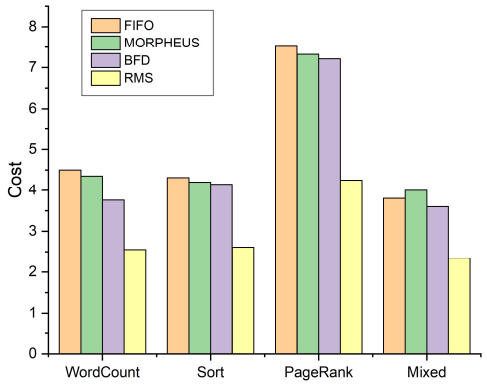

Figure. 3 Cost Efficiency at High Workload

The proposed RMS shows an efficient allocation of jobs and significantly reduces the cost of running jobs compared to other schedulers. The regular monitoring of resources and creating a RAV support RMS to schedule jobs efficiently in comparison to baseline schedulers. The FIFO scheduler shows the highest execution cost due to its creation of new pods for assigning jobs. Morpheus and BFD are able to maintain costs lower than FIFO due to their effective management of job orders in a queue that balances the cluster resources. Fig. 2 and 3 shows the cost efficiency of different scheduling procedures during low and high job workload

respectively. In this execution of low workload 4 Cores and 8 GB memory resources, and for high load 10 Core and 16 GB memory are utilized for execution of 4 different types of jobs.

In Fig. 2 the proposed RMS exhibits significant cost reduction during the low workload period due to effectively managing the jobs scheduling in the nodes. In comparison to the baseline scheduling algorithm, RMS reduces the cost of cluster node usage for WordCount, Sort, and Mixed applications by at least 33%, 39%, and 20%, respectively. For PageRank applications, RMS reduces resource consumption costs by at least 12% compared to FIFO. RMS also decreases resource utilization costs by 6% related to Morpheus. The RMS attempts to allocate the jobs in appropriate pods of a node with the support of RAV information, which improves job performance and thus reduces the overall cost of applications. In Fig. 3 during a high workload, the decrease in cost is lesser than low workload due to the overused of the cluster nodes. In this situation, RMS shows cost efficiency near 5% to 21% with variation in the job workloads.

B. Job Performance

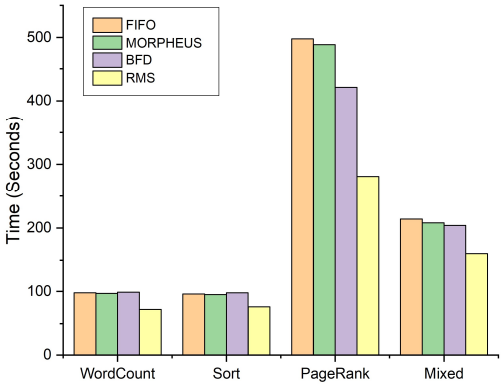

Figure. 4 Job Performance at Low Workload

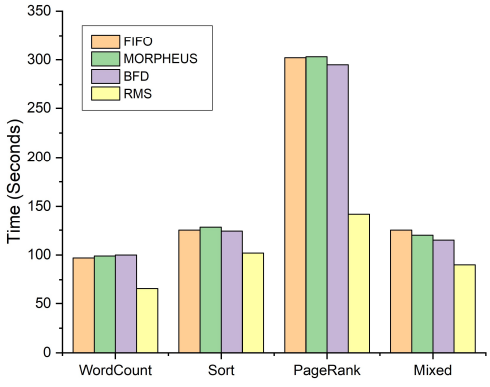

Figure. 5 Job Performance at High Workload

Fig. 4 and 5 show the average job finishing interval for the scheduling procedures at low and high workloads respectively. It shows RMS improvement over FIFO, Morpheus, and BFD with all four job types' execution. The enhancement of RMS is due to the use of RAV information for identifying best-fit pods within cluster nodes to accommodate the jobs by utilizing the demand resources to their maximum efficiency.

The result of the PageRank application of both FIFO and Morpheus shows the lowest job result because of extreme network communications when the exchange periods of the job allocation. However, RMS outperforms the comparing schedulers in both workload situations. It is because, in a low workload it smoothly placing of jobs due to prior resource availability information by the RM, and in a high workload the cluster gets overloaded due to not likely getting the required resource demand of the jobs. In such case, RMS reassign the jobs back in the queue and tries to avail the next best resource to assign job for execution. So, during a low workload, RMS improves runtime by at least 12%, and with a high workload 4%.

C. Scheduling Overhead

Scheduling overload among the baseline scheduler and proposed RMS is shown in Fig. 6. It measures the ratio of the total number of jobs allocated within the deadline given versus the total number of jobs waiting in queue for a time interval.

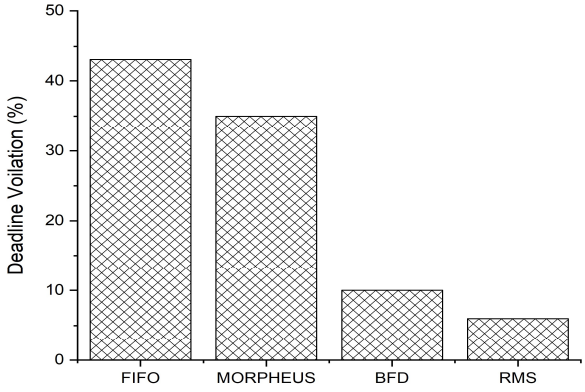

Figure. 6 Scheduling overload for schedulers

Fig. 6 represents the scheduling overload for different schedulers employing deadline violations. Here the higher the violation, the more overload in scheduling. FIFO shows the highest among all the schedulers as the demand of the resource might not be available during the request time due to which waiting time for jobs is and it leads to deadline violations. Morpheus independently determines the priority of work, where the work that leads is well-adjusted for the important job through sharing of resources in the cluster which makes it maintain lower violation than FIFO. The deadline violation between BFD and RMS shows a variation of 2%. As BFD uses a modest scheme known as “Earliest Deadline First”, where entire jobs are organized as per their deadlines, and for the new job the deadline is scheduled first. But, RMS shows the least among all as it allocates job much quicker due to the node resource availability information using RAV which make RMS select the best-fit node to place the jobs without violating the deadline. So, the enhancement in job performance and reduction in scheduling overhead in RMS with heterogeneous cluster nodes provides a better execution of jobs with high cost-effectiveness.

V. Conclusion

Job scheduling is being identified as a challenging task for big data applications due to its varying resource demand. In this paper, we present an RMS for efficient job scheduling through regular node resource monitoring. The RMS employs a resource monitoring mechanism for creating RAV for the cluster node which provides the information number of free Core and free memory size availability. The RMS mechanism will monitor the availability of various processes and resources in the scheduling process, leading to satisfaction with the use of these resources. Using RAV data it developed a scheduling algorithm called RB-BFS to improve resource management and improve cost-effectiveness by accurately deciding the allocation of the job as per their requirements. Extensive analysis against the baseline scheduler has been promising regarding the performance of RMS using reference application datasets in Spark and Kubernetes with a heterogeneous cluster node. The outcome results show an enhancement in

job performance and a reduction in scheduling overhead in RMS with heterogeneous cluster nodes providing a better execution of jobs with high cost-effectiveness. Further research could examine the suitability of this algorithm in other cloud scenarios, such as network resource, deadline sensitive and sensitive and priority jobs scheduling.

References

[1]. M. T. Islam, H. Wu, S. Karunasekera and R. Buyya, "SLA-Based Scheduling of Spark Jobs in Hybrid Cloud Computing Environments," IEEE Transactions on Computers, Vol. 71, no. 5, pp. 1117-1132, 2022.

[2]. Y. Huang, H. Xu, H. Gao, X. Ma, and W. Hussain, "Ssur: an approach to optimizing virtual machine allocation strategy based on user requirements for cloud datacentre", IEEE Transactions on Green Communications and Networking, Vol. 5, no. 2, pp. 670–681, 2021.

[3]. X. Ma, H. Xu, H. Gao, and M. Bian, "Real-time multiple workflow scheduling in cloud environments", IEEE Transactions on Network and Service Management, Vol. 18, no. 4, pp. 4002–4018, 2021.

[4]. S. Zhang, C. Wang and A. Y. Zomaya, "Robustness Analysis and Enhancement of Deep Reinforcement Learning-Based Schedulers," in IEEE Transactions on Parallel and Distributed Systems, Vol. 34, no. 1, pp. 346-357, 2023.

[5]. Y. Li, T. Li,P. Shen, L. Hao, J. Yang ,Z. Zhang, J. Chen, L. Bao, "PAS: Performance-Aware Job Scheduling for Big Data Processing Systems", Security and Communication Networks, Vol. 2022, ArticleID: 8598305, pp. 14, 2022.

[6]. R. Yang et al., "Performance-Aware Speculative Resource Oversubscription for Large-Scale Clusters", IEEE Transactions on Parallel and Distributed Systems, Vol. 31, no. 7, pp. 1499-1517, 2020.

[7]. J. Zhu, X. Li, R. Ruiz, W. Li, H. Huang and A. Y. Zomaya, “Scheduling Periodical Multi-Stage Jobs With Fuzziness to Elastic Cloud Resources”, IEEE Transactions on Parallel and Distributed Systems, Vol. 31, no. 12, pp. 2819-2833, 2020.

[8]. X. Zhou, W. Liang, K. Yan, W. Li, K. I-K. Wang, J. Ma, Q. Jin, "Edge-Enabled Two-Stage Scheduling Based on Deep Reinforcement Learning for Internet of Everything", IEEE Internet of Things Journal, Vol. 10, no. 4, pp. 3295-3304, 2023.

[9]. L. Zhu, K. Huang, Y. Hu and X. Tai, “A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata”, IEEE Access, Vol. 9, pp. 81236-81252, 2021.

[10]. X. Zhang, L. Li, Y. Wang, E. Chen and L. Shou, “Zeus: Improving Resource Efficiency via Workload Colocation for Massive Kubernetes Clusters", IEEE Access, Vol. 9, pp. 105192-105204, 2021.

[11]. A. Dina, A. Gamal, Z. Ibrahim, and A. N. Aida, “Elite learning Harris hawks optimizer for multi-objective task scheduling in cloud computing”, Journal of Supercomputing, Vol. 78, no. 2, pp. 2793–2818, 2022.

[12]. M. S. A. Khan and R. Santhosh, "Task scheduling in cloud computing using hybrid optimization algorithm", Soft Comput, Vol. 26, pp. 13069–13079, 2022.

[13]. V. Meyer, D. F. Kirchoff, L. Matheus, D. Silva, A. Cesar, and F. De Rose, "ML-driven classification scheme for dynamic interference-aware resource scheduling in cloud infrastructures", Journal of Systems Architecture, Vol. 116, Article ID: 102064, 2021.

[14]. R. Chen, X. Chen, and C. Yang, "Using a task dependency job scheduling method to make energy savings in a cloud computing environment", Journal of Supercomputing, Vol. 78, 2021.

[15]. F. Cheng, Y. Huang, B. Tanpure, P. Sawalani, L. Cheng, and C. Liu, "Cost-aware job scheduling for cloud instances using deep reinforcement learning", Cluster Computing, Vol. 25, no. 1, pp. 619–631, 2022.

[16]. Y. Fan, "Job scheduling in high performance computing", Distributed, Parallel, and Cluster Computing, Vol. 18, 2021.

[17]. B. Zheng, Li Pan, and S. Liu, "Market-oriented online bi-objective service scheduling for pleasingly parallel jobs with variable resources in cloud environments", Journal of Systems and Software, Vol. 176, ArticleID: 110934, 2021.

[18]. Y. Shao, C. Li, J. Gu, J. Zhang, and Y. Luo, "Efficient jobs scheduling approach for big data applications", Computers & Industrial Engineering, Vol. 117, pp. 249–261, 2018.

[19]. K. Chen and L. Huang, "Timely-throughput optimal scheduling with prediction", IEEE/ACM Transactions on Networking, Vol. 26, no. 6, pp. 2457–2470, 2018.

[20]. X. Hou, T. K. Ashwin Kumar, J. P. Thomas, and H. Liu, "Dynamic deadline-constraint scheduler for hadoop YARN", In Proceedings of the IEEE SmartWorld, Ubiquitous Intelligence& Computing, Advanced & Trusted Computed, Scalable Computing& Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, San Francisco, CA, USA, pp. 1–8, 2017.

[21]. Y. Wang and W. Shi, "Budget-driven scheduling algorithms for batches of Mapreduce jobs in heterogeneous clouds", IEEE Transactions on Cloud Computing, Vol. 2, no. 3, pp. 306–319, 2014.

[22]. Y. Yao, J. Wang, B. Sheng, J. Lin, and N. Mi, "Haste: Hadoop YARN scheduling based on task-dependency and resource demand", In Proceedings of the International Conference on Cloud Computing, Anchorage, AK, USA, pp. 184–191, 2014.

[23]. M. Khan, Y. Jin, M. Li, X. Yang, and C. Jiang, "Hadoop performance modeling for job estimation and resource provisioning", IEEE Transactions on Parallel and Distributed Systems, Vol. 27, no. 2, pp. 441–454, 2015.

[24]. Z. Niu, S. Tang, and B. He, "An adaptive efficiency-fairness meta-scheduler for data-intensive computing", IEEE Transactions on Services Computing, Vol. 12, 2016.

[25]. Q. Wang and X. Huang, "Pft: a performance-fairness scheduler on hadoop yarn", in Proceedings of the International Conference on Software Engineering and Service Science (ICSESS), pp. 76–80, IEEE, Beijing, China, August 2016.

[26]. M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling", In Proceedings of the 5th European conference on Computer systems, ACM, Paris, France, pp. 265– 278, 2010.

[27]. G. Ali, M. Zaharia, H. Benjamin, and A. Konwinski, "Dominant resource fairness: fair allocation of multiple resource types", USENIX Security Symposium, Vol. 11, p. 24, 2011.

[28]. S. Tang, B. S. Lee, and B. He, "Dynamicmr: a dynamic slot allocation optimization framework for mapreduce clusters", IEEE Transactions on Cloud Computing, Vol. 2, no. 3, pp. 333–347, 2014.

[29]. S. Selvarani and G. S. Sadhasivam, "Improved cost-based algorithm for task scheduling in cloud computing", In Proceedings of the International Conference on Computational Intelligence and Computing Research (ICCIC ’10), Coimbatore, India, pp. 1–5, 2010.

[30]. J. Li, M. Qiu, J. Niu, W. Gao, Z. Zong, and X. Qin, "Feedback dynamic algorithms for preemptable job scheduling in cloud systems", In Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT ’10), Toronto, Canada, Vol. 1, pp. 561–564, 2010.

[31]. Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. K. Tsang, "Rush: a robust scheduler to manage uncertain completion-times in shared clouds", In Proceedings of the International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, pp. 242–251, 2016.

[32]. Y. Yang, Y. Zhou, Z. Sun, and H. Cruickshank, "Heuristic scheduling algorithms for allocation of virtualized network and computing resources", Journal of Software Engineering and Applications, Vol. 6, no. 1, pp. 1–13, 2013.

[33]. S. Ghanbari and M. Othman, "A priority based job scheduling algorithm in cloud computing", Procedia Engineering, Vol. 50, pp. 778–785, 2012.

[34]. M. A. Sharkh, A. Ouda, A. Shami, “A resource scheduling model for cloud computing data centers”, In Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 213 - 218, 2013.

[35]. A. Nathani, S. Chaudhary, G. Somani, "Policy based resource allocation in IaaS cloud", Future Generation Computer Systems, Vol. 28, no. 1, pp. 94 - 103, 2012.

[36]. J. Sgall, "A new analysis of Best Fit bin packing", In Proc. of 6th Int. Conference FUN with Algorithms, Vol. 7288, pp. 315-321, 2012.

[37]. L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al., "Bigdatabench: A big data benchmark suite from internet services", In Proc. of the International Symposium on High Performance Computer Architecture (HPCA), 2014.

[38]. T. Islam, S. N. Sriramaa, S. Karunasekeraa, R. Buyya, "Cost-efficient dynamic scheduling of big data applications in apache spark on cloud", The Journal of Systems and Software, Vol. 162, ArticleID: 110515, 2020.

[39]. S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. N. Goiri, S. Krishnan, J. Kulkarni, S. Rao, "Morpheus: Towards automated slos for enterprise clusters", In Proc. of the 12th USENIX Conf. on Operating Systems Design and Imp. (OSDI), 2016.