International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Scheduling Scientific Workflow to Improve Service Quality Parameters in the Cloud Computing

[1]Prof. Riya Gohil, [2]Dr. Hiren Patel

Submitted: 08/05/2023 **Revised**: 15/07/2023 **Accepted**: 08/08/2023

Abstract: Cloud computing has emerged as a crucial platform for managing and executing time-constrained scientific applications, typically represented by workflow models and their scheduling. The scheduling of workflow applications in cloud computing poses a significant challenge, as they consist of numerous tasks with complex structures involving processing, data entry, storage access, and software functions. To address this challenge, users are provided with a convenient and cost-effective approach to run workflows on rented on-cloud Virtual Machines (VMs) at any time and from anywhere. With the growing dominance of pay-as-you-go pricing models in cloud services, extensive research has been conducted to minimize the cost of workflow execution by developing customized VM allocation mechanisms. However, most existing approaches assume static task execution times in the cloud, which can be estimated in advance. Unfortunately, this assumption is highly impractical in real-world scenarios due to performance variations among VMs. In this study, we propose a custom workflow scheduling algorithm designed to handle deadline-constrained workflows with random arrivals and uncertain task execution times, while ensuring higher CPU utilization. Our algorithm supports the use of containers to manage targets and optimize resource utilization, thereby reducing the overall cost of infrastructure resources and meeting individual workflow deadline constraints. Simulation results demonstrate that the proposed algorithm outperforms existing approaches in terms of rental costs and resource utilization efficiency.

***Index Terms -**: Scientific workflow, Cloud computing, Amazon Web Services, Scheduling*

1. Introduction

Cloud computing offers computing resources like CPU, memory, hard plates and data transmission and programming assets including virtual program and workflow programming [1, 2]. The Cloud supplier simply focuses on further developing the help capacities of the Cloud stage to address client issues and fulfill client requirement [3, 4].

Virtualization is one of the critical empowering innovations of Cloud computing which permits various VMs to reside on a solitary actual machine [7]. A VM copies a specific PC framework and exe-cutes the client's undertakings [8]. By utilizing the instantiation of the VMs, clients can send their applications on resources with different execution and cost levels. In each physical machine or server, the VMs are overseen by a product layer called hypervisor or the VM screen which works with the VM creation and scheduled execution.

Workflow booking is one of the conspicuous issues in Cloud computing which attempts to plan the work process assignments to the VMs in light of various utilitarian and non-useful requirements [9]. A workflow comprises a series of related undertakings which are limited together through data or utilitarian conditions and these conditions ought to be considered in the planning [10]. In any case, workflow scheduling in Cloud computing is a NP-hard optimization problem and accomplishing an ideal schedule is troublesome [11]. NP is a complexity class that represents the set of all decision problem for which the instances where the answer is “yes” have proofs that can be verified in polynomial time. This means that if someone gives us instances of the problem and a certificate to the answer being yes, we can check that it is correct in polynomial time. NP hard intuitively, these are the problems that are at least as hard as the NP complete problem. Note that NP hard problems do not have to be in NP, and they do not have to be decision problems. Because there are various VMs in a Cloud and numerous client undertakings ought to be scheduled by thinking about different auto scaling methods.

The normal target of the work process planning techniques is to limit the makespan by the appropriate assignment of the tasks to the virtual assets [12]. For instance, for instance preplanning of auto scaling techniques help to achieve promised SLAs, the client indicated cutoff times and cost constraints [13].Also, scheduling solutions may consider factors such as load balancing availability of the Cloud resources, resource utilization and services in the scheduling decisions.

The workflow planning issue has been broadly examined. This paper presents a total overview of the workflow scheduling algorithms. For this reason, it first sorts and objectives of workflow scheduling and afterward gives a classification of the proposed plans in light of the

[1]*LDRP-ITR Engineering college, Sarva Vidyalaya Kelavani Mandal, Gujarat, India and*
[2]*Principal, VSITR, Sarva Vidyalaya Kelavani Mandal, Gujarat, India*

calculation which has been utilized in every workflow scheduling plan. Likewise, the objectives, properties and the constraints of workflow scheduling schemes are surveyed exhaustively and a total correlation of them is introduced. Albeit a few plans, for example [14] examined the workflow scheduling problem in Cloud computing, is not the only solution, a top to bottom examination and correlation of the proposed workflow scheduling plans [15].

Common performance metrics for workflow scheduling algorithms include:

Makespan: This measures the total execution time of the workflow, from the start of the first task to the completion of the last task. A lower makespan indicates better performance.

Resource Utilization: This measures the utilization of the Cloud resources, such as CPU, memory, and storage. Higher resource utilization indicates better performance.

Cost: This measures the cost of executing the workflow on the Cloud platform. This includes the cost of using the Cloud infrastructure, data transfer costs, and any other related costs. A lower cost indicates better performance.

Deadlines Compliance: This measures the ability of the algorithm to meet the deadline constraints for each task in the workflow. A higher compliance rate indicates better performance.
Reliability: This measures the ability of the algorithm to perform consistently and accurately under different conditions and workloads.

Scalability: This measures the ability of the algorithm to handle increasing workloads and larger workflows without a significant decline in performance.

The use of Cloud computing for scientific applications has grown rapidly in recent years, due to its ability to provide on-demand access to computational resources. However, scheduling workflow applications in Cloud computing can be a complex and challenging task, given the dynamic nature of task execution times in Cloud environments. In addition, the increasing popularity of Cloud computing has led to a rise in pay-as-you-go pricing models, making it more important than ever to develop efficient and cost-effective solutions for scheduling workflows in the Cloud.

The objective of this work is to address the challenge of scheduling workflow applications in Cloud computing by proposing a novel algorithm that takes into account the dynamic nature of task execution times and aims to minimize the cost of executing workflows while maximizing resource utilization efficiency.

The main contribution of this work is the development of a custom scheduling algorithm for deadline-constrained workflows with random arrivals and uncertain task execution times. The algorithm supports the use of containers to manage resources and meet workflow deadlines, and it has been evaluated through simulation results, which show that it significantly outperforms existing solutions in terms of rental costs and resource utilization efficiency. This work makes a valuable contribution to the field of Cloud computing by offering a practical solution for scheduling workflows in Cloud environments that addresses the real-world challenges posed by dynamic task execution times and cost-sensitive pricing models."

2. The Implementation of Our Strategy

Cloudsim [15] simulation toolkit is the more generalized and effective simulator for testing Cloud computing related hypothesis. This toolkit allows seamless modeling, simulation and experimentation related to Cloud based infrastructures and application services. We created simulation on Cloudsim for shortest job first (SJF) scheduling and identified how the Cloudsim simulation works for scheduling process. Cloudsim simulation is not given accurate result for scientific workflow application. For scientific workflow application the other Cloud platform like Amazon Web Services (AWS), Azure, Google Cloud etc. are provide more accurate result [16].Instead of simulation now we moved on the real Cloud where we used AWS. In the empirical part of study, we validated that AWS Elastic Cloud (EC2) [19] can be used for our proposed plan for scientific workflow scheduling in Cloud computing for enhancing QoS parameters like cost, makespan and budget.

AWS offers feature such as autoscalling and load balancing witch motivated us to use it for the purpose of scientific workflow scheduling. The default AWS scheduler schedules the task within built algorithm (e.g. SJF), and it permits the developer to replace SJF with their algorithm for testing purpose [17,18].

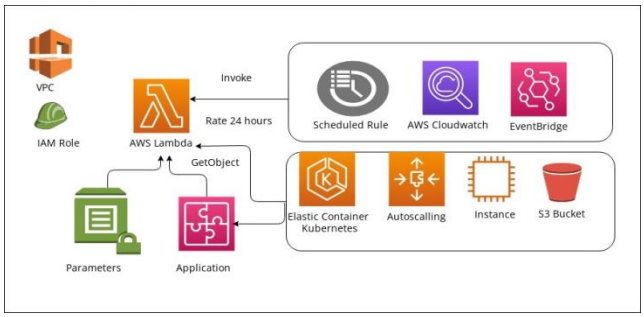

Fig. 1 System Architecture[19]

Fig. 1 shows AWS architecture with customized configuration system. Amazon Virtual Private Cloud (VPC) enables to launch AWS resources into a virtual network that have been defined. This virtual network closely resembles a traditional network that you'd operate

in your own data centre, with the benefits of using the scalable infrastructure of AWS [17]. Identity and Access Management (IMA) roles allow you to delegate access to users or services that normally don't have access to your organization's AWS resources. IAM used to control access to AWS resources, like which users are authenticated and who is authorized. IAM users or AWS services can assume a role to obtain temporary security credentials that can be used to make AWS API calls. AWS Lambda [18] is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. One can trigger Lambda from over 200 AWS services and software as a service (SaaS) applications, and only pay for what you use.

Amazon *CloudWatch* [19] is a monitoring and management service that provides data and actionable insights for AWS, hybrid, and on-premises applications and infrastructure resources. You can collect and access all your performance and operational data in the form of logs and metrics from a single platform rather than monitoring them in silos (server, network, or database). *CloudWatch* enables you to monitor your complete stack (applications, infrastructure, and services) and use alarms, logs, and events data to take automated actions and reduce mean time to resolution (MTTR). Amazon Elastic *Kubernetes* Service (Amazon EKS) is a managed *Kubernetes* [19] service that makes it easy for you to run *Kubernetes* on AWS and on-premises. *Kubernetes* is an open-source system for automating deployment, scaling, and management of containerized applications.

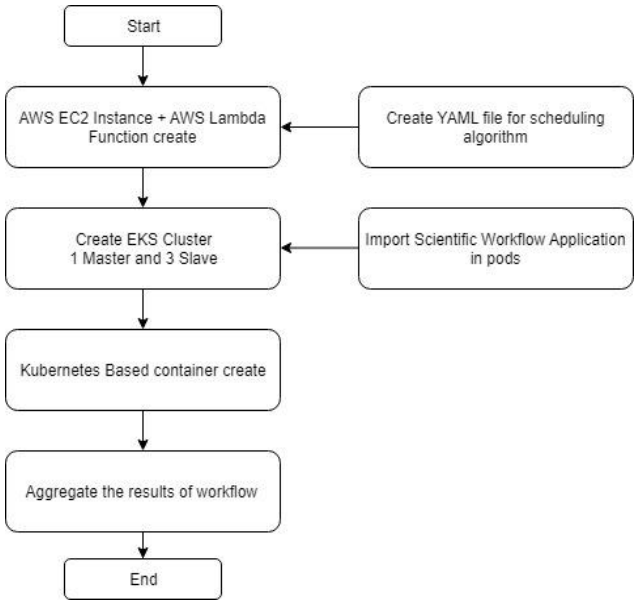

Fig. 2 Process Flowchart

Fig. 2 defines how our scheduler works with LAMBDA and EC2 instances. An EKS cluster [20] has been created which takes place as a 1 master node and 3 slave nodes have also been created. We have applied a YAML file [21] for scheduling based on CPU. It is based on containerized service and now the next step is to import scientific workflow applications to aggregate the results with the usage of scientific application. As we are using a scientific application, there is a large amount of data in that application : We are going to check with AWS Cloud watch service and also monitor the load with the configuration with Cloudwatch.

3. Scheduling Algorithm

Kube-scheduler[kubelet][18,22] is first decide which node to place with pod. After specifying the resource limit of container, kubelet apply those limits. The running container is not allowed to use more resources based on limit set. Second, kubelet reserves least request amount of that system specifically which containers are in used.

Containers are managed by a kubelet. It is responsible for starting, stopping and restarting containers. Each time a pod is requested by a client application, it passes information about its resources required (e.g. CPU and memory) and limits (e.g. request and limit). The kubelet checks this information against its own configuration and decides on how many CPUs or memory units are needed. It then selects an appropriate host machine where these resources can be allocated.Thekubelet manages allocating CPU resources by monitoring load on each host machine where containers are running. It will try to start one new container if there are no enough resources available. This happens automatically when there are too few resources available at any point in time.

Below is the algorithm which we have used for scheduling:

As the title of the section is Scheduling Algorithm, It should talk more about our proposed algorithm, its strengths, comparison with existing algorithms etc. The algorithm proposed scientific workflow scheduling_Resource Allocation (SWFS_RA) in the paper is to be a custom scheduling algorithm for deadline-constrained workflows with random arrivals and uncertain task execution times in a Cloud computing environment. The algorithm aims to minimize the cost of executing workflows while maximizing resource utilization efficiency.

Algorithm Parameters are defined like:-Batch size (B): The number of jobs that are processed in each instance. This parameter determines the number of jobs that are scheduled and executed simultaneously. Number of instances (L): The number of virtual machines (VMs) required executing the workflow. This parameter affects the resource utilization and overall cost of executing the workflow. Resource utilization: The utilization of the available resources (e.g. CPU, memory, storage) by the instances. This parameter affects the performance and cost of executing the workflow. Task execution time (T_i and

T_j): The time required to complete each task in the workflow. This parameter affects the overall execution time of the workflow. LoadBalancing array (LoadBalancing[m]): An array that counts the number of sent jobs to each instance. This parameter determines the distribution of the workload among the instances.

Here is a brief overview of the steps involved in the algorithm:Ti and Tj are defined as the successor and dependent jobs, respectively. The number of scheduled jobs is defined as jobsnum. The "L" variable represents a lambda instance. The batch size is defined as the number of jobs in a single lambda instance. The alertmax variable is set to true if the number of scheduled jobs is equal to the batch size. The "for" loop runs from 1 to "q" and performs the following steps:

a. jobsnum is incremented by 1.

b. If jobsnum is equal to the batch size, alertmax is set to true and the loop is broken.

If alertmax is set to true, the loadbalancing array is updated to reflect the number of sent jobs.

Table1. Symbol and Abbreviation for SWFS_RA Algorithm

Symbol	***Abbreviations***
Ti	*The execution time of task i.*
Tj	*The execution time of task j.*
L	*The number of instances (i.e. VMs) required executing the workflow.*
B	*The batch size of jobs in each instance.*
N	*The number of jobs in the workflow.*
M	*The number of instances in the Cloud environment.*
J_num	*The number of scheduled jobs.*
LoadBalancing[m]	*An array that counts the number of sent jobs to each instance.*

For each request, the algorithm checks if the group is a CPU group, and then perform the following checks:

a. Check if a container is available in the pod.
b. Check if the container is of the CPU group.
c. Check if sufficient resources are available.
d. Check if there is a CPU group request.
e. If all checks pass, launch a new pod or container in the same pod, depending on the situation.

The algorithm is designed to handle CPU-intensive tasks in a Cloud computing environment, and to make efficient use of available resources to minimize costs and maximize resource utilization. Table 1 represents the symbol and their abbreviations used in proposed algorithm.

In this mathematical model, the algorithm first increments the number of scheduled jobs (J_num) until it reaches the batch size (B). Once the batch size is reached, the number of sent jobs to each instance is updated in the Load Balancing array. For each request, the algorithm performs a series

of checks to determine whether a new pod should be launched, a container should be launched in the same pod, or a container should be launched in a different pod. The checks include verifying the existence of a CPU group request, checking the availability of sufficient resources, and checking the type of group (i.e. CPU or non-CPU). Algorithm: SWFS_RA

Step1: Define the variables:

Ti and Tj as previously defined.

L, B, N, M, jobsnum, and LoadBalancing[m] are the same as the algorithm.

S_ij, C_m, P_ijm, X_im, and Y_m as defined in the mathematical model.

Step 2: For j = 1 to N, perform the following steps:

Increment jobsnum by 1.

Check if jobsnum is equal to B.

If true, set alertmax to true and break out of the loop.

If alertmax is true, update LoadBalancing[m] to reflect the number of sent jobs.

Step 3: For each request, perform the following steps:

Check if the group is a CPU group.

If true, check if a container is available in the pod.

If a container is available, check if it is of the CPU group.

If it is of the CPU group, check if there are sufficient resources available and if there is a CPU group request.

If all checks pass, launch a new pod or container in the same pod, depending on the situation.

Step 4: Define the objective function:

Minimize the sum of the communication costs between tasks i and j, multiplied by the binary variable indicating whether tasks i and j are executed on the same instance.

Step 5: Define the constraints:

The execution time of task i multiplied by the binary variable indicating whether task i is executed on instance m should be less than or equal to the capacity of instance m, for all m.

The sum of binary variables indicating whether task i is executed on instance m should be equal to 1, for all i.

The binary variable indicating whether instance m is used or not should be greater than or equal to the binary variable indicating whether task i and task j are executed on the same instance, for all i, j.

SWFS_RA algorithm is performing load balancing in a cluster of some kind. It iterates over a number of requests, represented by the variable N, and keeps track of how many requests have been processed using the variable J_num. When J_num reaches a certain threshold, represented by the variable B, the variable alertmax is set to true and the loop is broken. After the loop, if alertmax is true, the algorithm updates a load balancing variable for each node in the cluster, represented by the variable LoadBalancing [m]. It then enters a loop over each request and performs some actions based on the group of the request. If the group of the request is CPU, the algorithm checks whether the container is already in a pod and whether the container is in the CPU group. If it is, it checks whether there are sufficient resources available for the CPU group request and whether a CPU group request already exists. If both conditions are true, it launches a new pod for the request. Otherwise, it launches the container in the same pod. If the container is not in a pod or not in the CPU group, the algorithm launches a new pod for the request.

The algorithm checks whether a given request belongs to the CPU group, and then checks whether there is a container for that request already running in a pod. If there is, the algorithm checks whether that container belongs to the CPU group and whether there are sufficient resources available for the request. If there are, the algorithm launches a new pod for the request. If there is no container for the request, or if the container does not belong to the CPU group, the algorithm launches a new pod for the request. In general, a scheduler is a program that assigns tasks or resources to different entities in a system according to some set of rules or criteria. This algorithm is a scheduler for CPU resources

SWFS_RA is a load balancing and resource allocation system for a cluster of nodes that are used to run containers. Such systems are used to ensure that resources are used efficiently and effectively, and to prevent any individual node or container from becoming overloaded. Load balancing is the process of distributing workloads across multiple resources in order to optimize performance and avoid overloading any one resource. In the context of a container cluster, load balancing typically involves distributing containers across multiple nodes so that each node is utilized evenly. Resource allocation is the process of assigning resources to specific tasks or processes in order to ensure that those tasks or processes have sufficient resources to complete their work. In the context of a container cluster, resource allocation typically involves ensuring that each container has sufficient resources to operate correctly, such as CPU, memory, and storage. By using a load balancing and resource allocation system such as the algorithm provided, organizations can ensure that their container clusters are operating efficiently and effectively, with each container receiving the resources it needs to operate correctly, and with workloads being distributed evenly across the cluster. This can help to prevent system failures, reduce downtime, and improve overall system performance.

4. Experiments and Results

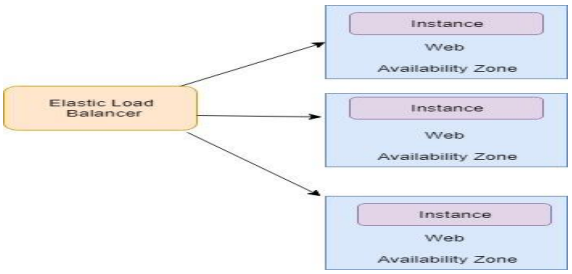

Fig. 3 ELB Diagram

Fig. 3 ELB Diagram is representing Elastic load balancer. Load balancer is a way to balance the traffic by sending client request to multiple backend servers. Also load balancer checks the health of backend servers.AWS Lambda [23] is a serverless computing administration given by Amazon Web Services (AWS). Clients of AWS Lambda make capabilities, independent applications written in one of the upheld dialects and runtimes, and transfer them to AWS Lambda, which executes those capabilities in a proficient and adaptable way.The Lambda capabilities can play out any sort of computing task, from serving site pages and handling floods of information to calling APIs and coordinating with other AWS administrations.

The idea of "serverless" computing alludes to not expecting to keep up with your own servers to run these capabilities. AWS Lambda is a completely overseen administration that deals with all the framework for you. Thus "serverless" doesn't intend that there are no servers included: it simply implies that the servers, the working frameworks, the organization layer and the remainder of the foundation

have previously been dealt with, so you can zero in on composing application code.

Amazon CloudWatch serves as the monitoring service within AWS, responsible for gathering metrics and logs from your resources. Within CloudWatch, there exists a sub-service called Amazon CloudWatch Events, which promptly streams system events in near-real time whenever changes occur in your AWS resources. By defining targets for these events, we can take appropriate actions. For instance, when an EC2 instance is started, it sends an event to Amazon CloudWatch Events. We can create a rule that triggers an AWS Lambda function whenever this event occurs. These events are triggered in response to changes in your AWS resources. We have the capability to define event rules that self-trigger at regular intervals, along with configuring a target action to perform routine tasks. We designate an Amazon Lambda function as the scheduled target. Once the specified time or interval for this event is reached, our function is executed. These types of events are referred to as scheduled Amazon CloudWatch Events.

We define event rules that self-trigger regularly and configure a target action to do some regular work. So we define an Amazon Lambda function as scheduled targets. when this event is triggered at the specified time or interval you defined, our function is executed. These types of events are called scheduled Amazon CloudWatch Events.

Amazon EventBridge, the serverless event bus service, is the upgraded version of CloudWatch Events. It simplifies the creation of an event-driven architecture and facilitates the integration of SaaS applications with AWS resources. To create scheduled events, we utilized the Amazon EventBridge console. The Horizontal Pod Autoscaler (HPA) continuously monitors metric threshold values that we have configured. By default, HPA checks these values every 15 seconds, but it can be adjusted using the --horizontal-pod-auto-scaler-sync-period flag according to our needs. If the current threshold exceeds the specified threshold, HPA attempts to increase the number of pods. The HPA controller assumes a linear relationship between the metric and the number of pods, operating based on the ratio between the desired metric value and the current metric value.The formula used to compute the desired replicas is as follows.

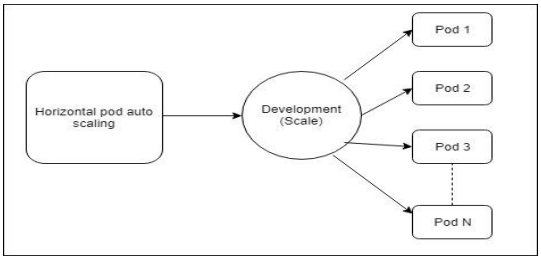

Fig. 4 Horizontal Pod Auto scaling

Initially, the pod count =1 (this is the minimum number of pods specified) in the Horizontal Pod Autoscaler configuration. As the load increases, the pod count increases to 4, to maintain the CPU utilization below 50%.

Reasons to Scale Horizontally

Handle More Throughput

When we have very less traffic in our application or websites it can easily run with a single web server. But what about if we have a popular site and a large amount of traffic needs to be handled? Then it's not able to handle traffic with a single web server so we required more web servers to handle this traffic.

Fault Tolerance

Fault tolerance is frequently sought after by individuals who wish to enhance a system's resilience. To achieve better resilience in the face of failures, the approach often involves increasing the number of nodes, which is commonly referred to as "high availability" or "HA."

Need More Resources than You Can Get from a Single Node

When adopting a vertical scaling strategy, there may come a point where the resources available from a single node become insufficient. It becomes impossible to add more memory, disk space, or other components. At that juncture, it becomes necessary to explore horizontal scaling options to address the issue.

Determining the desired number of replicas
In this experiment, our focus is on horizontal pod auto-scaling, and we will scale the system based on CPU utilization, a widely used metric. It's worth noting that higher CPU utilization corresponds to increased latency. By maintaining lower levels of CPU utilization, we can also keep the application's latency at lower levels. The graph below illustrates the variation in CPU utilization for an I/O-bound microservice.

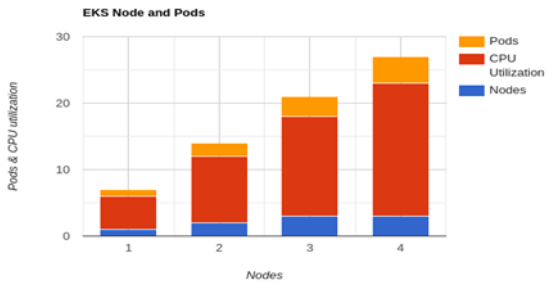

Fig. 5 experimental Results with CPU Utilization

Here in the Fig.5 experimental result with CPU Utilization its shows that with the three nodes initially we have one pod per node. After doing load testing on pods CPU utilization increased eventually nodes with pods value also

increased. These showcased that horizontal pod autos calling is working with hpa value increased. Here while doing experiment we have defines hpa value to 80 percentages. so whenever hpa crossed pods will increased that we can see in above graph. Horizontal pod auto scaling update the workload with the aim of auto scaling to match the demand of load.

5. Conclusions

With the fast advancement of the containerization method, CaaS is becoming increasingly more famous in Cloud computing administrations. For software engineering research centers in colleges, the asset is moderately restricted and the kinds of administrations are assorted. Workflow scheduling algorithm is one of the scheduling algorithms in load balancing. It is used to reduce makespan, cost, energy consumption, execution time and maximize resource utilization. Most of the researchers worked on important performance parameters such as makespan and cost, execution time, energy consumption and resource utilization. Our objectives on offering novel workflow scheduling framework are makespan, cost and reliability.

Firstly, we will make the scheduling algorithms more automated and stable by repairing the above defects. In addition, we will continue to research preprocessing of the dataset. We plan to do some practical analysis on workflow scheduling techniques. How AWS load balancing and instance scheduler used for our proposed plan and will implement it.

References

[1] Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J. An updated performance comparison of virtual machines and Linux containers. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015.

[2] Jennings, B.; Stadler, R. Resource Management in Clouds: Survey and Research Challenges. J. Netw. Syst. Manag. 2015, 23, 567–619. [CrossRef]

[3] Liu, P.; Hu, L.; Xu, H.; Shi, Z.; Tang, Y. A Toolset for Detecting Containerized Application's Dependencies in CaaS Clouds. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018.

[4] Dragoni, N.; Giallorenzo, S.; Lluch-Lafuente, A.; Mazzara, M.; Montesi, F.; Mustafin, R.; Safina, L. Microservices: Yesterday, today, and tomorrow. In Present and Ulterior Software Engineering; Springer: Cham, Switzerland, 2017.

[5] Singh, V.; Peddoju, S.K. Container-based microservice architecture for Cloud applications. In Proceedings of the 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, 5–6 May 2017; pp. 847–852. [CrossRef]

[6] A Survey on Scheduling Strategies for Workflow in Cloud Environment and Emerging Trends,Mainak Adhikari, Tarachand Amgoth and Satish Narayana Srirama,ACM, 2019.

[7] A GSA based hybrid algorithm for bi objective workflow scheduling in Cloud computing, Anubhav Choudhary, Indrajeet Gupta, Vishakha Singh and Prasanta K. Jana, FGCS, 2018.

[8] Minimizing cost and Makespan for workflow scheduling in Cloud using fuzzy dominance sort based HEFT, Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei and Shiyan Hu, Future Generation Computer System, 2018.

[9] GRP-HEFT :A Budget Constrained Resource Provisioning Scheme for Workflow Scheduling in IaaS Clouds, Hamid Reza Faragardi, Mohammad Reza Saleh Sedghpour, Saber Fazliahmadi, Thomas Fahringer and Nayereh Rasouli, IEEE, 2019.

[10] A Workflow Scheduling Deadline based Heuristic for Energy optimization in Cloud, Emile Cadorel, HeieneCoullon and Jean Marc Manaud, IEEE, 2019.

[11] Cost-Efficient and Latency-Aware Workflow Scheduling Policy for Container-based Systems, Weiwen Zhang, Yong Liu, Long Wang, Zengxiang Li and Rick Siow Mong Goh, IEEE, 2018.

[12] Concurrent Workflow Budget- and Deadline-constrained Scheduling in Heterogeneous Distributed Environments, N. Zhou, F. Li, K. Xu, and D. Qi, IEEE 2018.

[13] Understanding the performance and potential of Cloud computing for scientific application,ISadooghi ae al, IEEE, 2017.

[14] Makespan Driven Workflow Scheduling in Clouds Using Immune Based PSO Algorithm, Pengwei Wang, Yinghui Lei, Promise Ricardo Agbedanu, and Zhaohui Zhang, IEEE, 2020.

[15] Cost and Makespan aware workflow scheduling in hybrid Clouds, Junlong Zhou, Tian Wang, Peijin Cong, Pingping Lu, Tongquan Wei and Mingsong Chen, JSA 2019.

[16] Resource Provisioning for Task-batch based Workflows with Deadlines in Public Clouds, Z. Cai, X. Li, and R. Ruiz, ,IEEE 2019.

[17] http://Cloudbus.org/Cloudsim.

[18] "Uncertainty-Aware Online Scheduling for Real-Time Workflows in Cloud Service Environment, H. Chen, X. Zhu, G. Liu, andW. Pedrycz , IEEE 2018.

[19] https://aws.amazon.com/ec2/.

[20] “Cloud Pricing Models: Taxonomy, Survey, and Interdisciplinary Challenges, C. Wu, R. Buyya, and K. Ramamohanarao ACM, 2019.

[21] Cost and Makespan aware workflow scheduling in hybrid Clouds, Junlong Zhou, Tian Wang, Peijin Cong, Pingping Lu, Tongquan Wei and Mingsong Chen, JSA 2019.

[22] Multi Objective Cloud workflow scheduling: A Multiple Population Ant Colony System Approach, Zong-Gan Chen,Zhi Hui zhan, Yue Jiao Gong, Tian Long Gu, Feng Zhao, Hua Qiang Yuan, Xiaofeng Chen, Qing Li and Jun Zhang, IEEE 2018.

[23] Decomposition Based Multi Objective Workflow Scheduling for Cloud Environments,EmmanuelBugingo, Wei Zheng, Dongzhan Zhang, Yingsheng Qin and Defu Zhang, Easychair, 2019.

[24] Workflow Scheduling in Cloud computing environment with classification on ordinal optimization on using SVM, VahebSamandi, Debajyoti Mukhopadhyay and Nikhil Raut, IEEE 2019.

[25] Budget and Deadline Aware E Science Workflow Scheduling in Clouds, Vahid Arabnejad, Kris Bubendorfer and Bryan Ng, IEEE 2018.

[26] Dynamic Fault Tolerant Workflow Scheduling with Hybrid Spatial Temporal Re execution in Cloud, Na Wu, Decheng Zuo and Zhan Zhang, Information, 2019.

[27] A new Optimization Method for Security Constrained Workflow Scheduling,Ali Abdali and SafaMeasoomyNia, IJCSE, 2019.

[28] Fault Tolerant Scheduling for Scientific Workflow with Task Replication Method in Cloud, Zhongjin Li, Jiacheng Yu, Haiyang Hu, Jie Chen, Hua Hu, Jidong Ge and Victor Chang, Scipress,2018.

[29] Workflow Scheduling Using Hybrid GA-PSO Algorithm in Cloud Computing,Ahmad M. Manasrah and Hanan Ba Ali, Wiley, 2018.

[30] A Cuckoo based Workflow Scheduling Algorithm to Reduce Cost and Increase Load Balance in the Cloud Environment, Shahin Ghasemi and Ali Hanani, IJIV,2019.

[31] Workflow Scheduling in Cloud Computing Using Memetic Algorithm, Abdulsalam Alsmady,Tareq Al Khraishi, Wail Mardidni ,Hadeel Alazzam and Yaser Khamayseh, IEEE 2019.

[32] Performance Modeling and Workflow Scheduling of Microservice Based Application in Clouds, Liang Bao, Xiaoxuan Bu, Nana Ren and Mengqing Shen, IEEE, 2019.

[33] Workflow Scheduling Algorithm in Cloud Cmoputing,SavioVaz,Alisha Crystal D’Almeda and Santhosh B,IJERT 2019.

[34] User Priority Aware and Cost Constrained Workflow Scheduling in Clouds, Yuehing Chen, Yaunqing Xia, Ce Yan and Runze Gao, Chinese Control Conference, 2019.

[35] A QoS aware Workflow Scheduling Method for Cloudlet based Mobile Cloud Computing,Wei Tian, Renhao Gu, Feng Ruan, Xihua Liu and Shucun Fu, IEEE, 2019.

[36] Patil, S. D. ., & Deore, P. J. . (2023). Machine Learning Approach for Comparative Analysis of De-Noising Techniques in Ultrasound Images of Ovarian Tumors. International Journal on Recent and Innovation Trends in Computing and Communication, 11(2s), 230–236. https://doi.org/10.17762/ijritcc.v11i2s.6087

[37] Martinez, M., Davies, C., Garcia, J., Castro, J., & Martinez, J. Machine Learning-Enabled Quality Control in Engineering Manufacturing. Kuwait Journal of Machine Learning, 1(2). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/122

[38] Timande, S., & Dhabliya, D. (2019). Designing multi-cloud server for scalable and secure sharing over web. International Journal of Psychosocial Rehabilitation, 23(5), 835-841. doi:10.37200/IJPR/V23I5/PR190698