Journal of Computer Sciences

Original Research Paper

Cost-Efficient Deadline Constrained Scientific Workflow Scheduling in Infrastructure-as-a-Service Clouds by Disqualifying Tasks with Anomalies

[1]Anver Shahabdeen Rahumath, [1]Santhosh Rajendran, [1]Natarajan Mohanasundaram and [2]Abdul Rahiman Malangai

[1]Department of Computer Science and Engineering, Karpagam Academy of Higher Education, India
[2]LBS Centre for Science and Technology, India

Article history
Received: 06-01-2022
Revised: 25-04-2022
Accepted: 27-04-2022

Corresponding Author:
Anver Shahabdeen Rahumath
Department of Computer Science and Engineering, Karpagam Academy of Higher Education, India
Email: anversr@gmail.com

Abstract: Cloud computing has transformed the way businesses and consumers think about their data and businesses. As a result, cloud computing is described as the on-demand availability of all computer system resources via the Internet as a paid service. Enhancing security is a major problem in cloud computing, which is also a major research topic because data is stored and processed in remote locations held by third parties. Another key research subject is the allocation of virtual machines to incoming workloads to decrease the cost of consumers' workload execution in cloud environments. Both of the aforementioned difficulties are addressed in this study. Deadline-constrained workflows are submitted to the application server, which goes through a pre-processing step, that identifies the presence of anomalies in the workflow tasks and disqualifies those tasks with anomalies, and schedules the adjusted workloads into heterogeneous Virtual machines using a modified-PCP algorithm. Our approach is compared to the IC-LOSS and IC-PCP algorithms that are already in use. In comparison to the existing IC-PCP and IC-LOSS algorithms, experimental results suggest that using the modified-PCP algorithm for deadline constrained workflows after deleting those anomalous tasks produces better results.

Keywords: Anomaly, Cloud Computing, Deadline, Partial Critical Path, Scientific Application Workflows, Scheduling

Introduction

Cloud computing enables pervasive computing which provides on-demand access to the programmable class of computing resources. Cloud computing has also gained notoriety as a result of its ability to provide processing, storage, and software-based services over the Internet. Security, on the other hand, remains a major worry and a significant roadblock to the adoption of the cloud model (Mehraj and Banday, 2021). One of the most significant ideas in data analysis is anomaly detection. If an information object deviates dramatically from usual data behavior in some sphere, it is classified as an anomaly. In general, it denotes that an object is distinct from the rest in a data array. It's crucial to detect these objects so that you can look at them from a different perspective and apply alternative detection methods (Hu *et al*., 2017).

Anomaly detection is founded on the premise that common behavior is more likely to be correct and that uncommon deviation from the norm (so-called anomalies) are more likely to be incorrect (Körber *et al*., 2021). In a Distributed Cloud Environment, most cloud apps, whether elastic or non-elastic in nature, execute their duties utilizing Virtual Machines (VM). When compared to non-elastic apps, most elastic applications are intended to run for long periods and fully utilize their resources. Intercommunicating VMs and/or non-intercommunicating VMs can make up elastic application requests (Sridharan and Domnic, 2021).

Modern computational and data-intensive science is increasingly reliant on scientific procedures. Scientific workflows have evolved as a versatile way to express large programs with data and control dependencies declaratively. Networked clouds are an appealing platform for deploying and executing research workflows due to the intrinsic flexibility of scientific processes, i.e., growing resource needs as they execute. Advanced virtualization technologies now allow research workflows

to be packaged in such a way that they are highly portable, predictable, high-performance, and performance isolated (Gaikwad *et al*., 2016).

Although modern scientific experiments are conducted on complex, largescale, distributed high-performance systems that are designed with reliability in mind (Snir *et al*., 2014), they can experience anomalies ranging from minor (e.g., network performance degradation) to major (e.g., file system integrity errors) (Ewa *et al*., 2017), affecting the performance of the applications that leverage their resources and increasing the chances of failure. When a Scientific application workflow is scheduled to the cloud and some of the activities in the workflow exhibit anomalous behavior, the expensive cloud resources are squandered, raising the customer's cloud usage costs.

For a scenario in which a cloud client submits his or her scientific workflow application to be executed and controlled by a workflow execution service with a turnaround time (deadline) to be met, we explore a scheduler modeled as an Integer Linear Programme (ILP) (van Zelst *et al*., 2018).

Software-as-a-Service (SaaS) and platform-as-a-service are two examples of workflow execution service owners (PaaS). Aside from being responsible for creating a plan that meets the workflow's deadline, the scheduler's goal is to reduce the monetary costs of workflow execution, allowing the SaaS/PaaS to maximize profit by lowering client expenses (Genez *et al.*, 2020; Tiwari and Garg, 2021).

Mainstream cloud computing systems can be divided into three types based on provisioning strategies and architectural patterns: Infrastructure clouds (IaaS), Platform clouds (PaaS), and software clouds (SaaS). The IaaS concept allocates resources to Virtual Machines (VMs) that are built-in data centers or server nodes (Pan *et al*., 2020; Osypanka and Nawrocki, 2020; Anshu *et al*., 2020).

Overprovisioning can be avoided by optimizing VM placement based on the VMs' resource demands rather than their requests. Resource over commitment (Dabbagh *et al*., 2015) is a technique that allows multiple VMs to be placed (or consolidated) on the same PM by sharing hardware resources beyond their actual capacity. Despite these advantages, data center owners are wary of consolidating their facilities. One method involves allocating VMs based on their resource needs (i.e., CPU, memory, and disc) so that the total demand is less than the PMs' resource capacity. However, because IaaS clients tend to overestimate their VM resource requests to guarantee that their application requirements are met at all times, this suffers from over-provisioning. As a result, the consolidated data center has a poor utilization rate and PMs are underutilized. Unfortunately, overcommitting can harm application performance by clogging up limited PM resources and resulting in large Quality of Service (QoS) violations and penalties (Torre *et al*., 2020). When anomalies in tasks are discovered, the anomalous tasks are removed from workflows and the remaining workflows are scheduled to IaaS Clouds using a modified version of the PCP algorithm. The following are our major contributions to this project:

- All the submitted workflows pass through a pre-processing step which detects the anomalies in the workflows, if any, using the following steps
- Uses a modified matrix profile to operate on data in real-time to detect the anomalies
- Uses an adaptive training method to reduce the false alarm rates due to the presence of the same type of anomalies that occurred
- The workflows after disqualifying the anomalous tasks are scheduled to IaaS clouds using a modified PCP algorithm
- The results are compared with existing IC-PCP algorithm and IC-LOSS algorithms that schedule the workflows without disqualifying the anomalous tasks in the workflows

Related Work

Anomaly Detection in Scientific Workflows

Rodriguez *et al*. (2018) describe a framework for detecting anomalies in Scientific Workflows that employs Hierarchical Temporal Memory (HTM), an unsupervised model that learns incrementally, to detect anomalies in the flow of resource consumption time-series data. (Somani *et al*., 2017) and (Rodriguez and Buyya, 2018) propose a taxonomy of several Distributed Denial-of-Service (DDoS) assaults for multi-tenant, Cloud-based infrastructures, which differ from traditional fixed on-premise infrastructures. Rather than a single target server, numerous stakeholders are involved in DDoS attacks against Cloud Environments. DDoS attacks on Cloud Environment have an impact on many parameters other than those planned by the attackers. DDoS assaults also have an impact on other aspects such as the economy, business, and overall energy use. The report outlines measures to reduce the effects of the aforementioned variables.

Chen *et al*. (2018) proposed a defense-in-depth protection framework for tenant VMs to meet the VM security requirements concerning the access control of communications, anomaly detection in networks, monitoring of memory, as well as antivirus in files in an IaaS platform, which used three layers to meet the mentioned security concerns of customer trade from exterior to the interior of VM. A tenant territory model was conceptualized and implemented at the initial layer, restoring the ability to manage the communication

approach for VM service and maintaining security confinement of distinct tenant trade networks using Software Defined Networking (SDN). Alguliyev *et al*. (2019) provide a semi-supervised classification strategy for identifying anomalies in cloud infrastructure performance metrics based on a combination of classifiers. The suggested approach for generating ensemble Naive Bayes uses the SMO, J48, IBK, multilayer perceptron as well as PART algorithms. Using public data from Yahoo and Google, the MATLAB, Weka, Python 2.7, and SDK Shell programs were utilized to discover odd behavior on performance indicators. As a result of this research, a conclusion was reached that the detection accuracy of the model is 90%.

Scheduling in Cloud

The term Scheduling refers to the method of binding a task or a series of tasks to a collection of cloud virtual machines to meet the requirements of the users. The two terminologies related to cloud computation are task scheduling and workflow scheduling and the only difference between them is the range of data involved in the computation. A task is a single job that includes customer applications. A workflow, on the other hand, denotes large business or scientific data patterns, which consists of the number of interrelated tasks, usually represented as a Directed Acyclic Graph (DAG). A cybershake workflow Fig. 1 (WG) is an example of workflow DAG, which is used by the Southern Calfornia Earthquake Center to characterize earthquake hazards in a region, which consists of many tasks categorized into five broad categories. Scheduling in the cloud can improve the utilization of bought cloud resources, resulting in a better price-performance ratio. The majority of cloud scheduling concerns Infrastructure as a Service (IaaS) clouds.

The Scheduling Algorithm is one of the most important factors to consider when scheduling apps in a cloud environment (Varshney and Simmhan, 2020). The mapping strategy and type of techniques utilized are used to categorize the scheduling algorithms. Heuristics and meta-heuristics techniques are the general techniques used here. Greedy, brute-force, dynamic programming, and divide and conquer are examples of heuristics strategies. Meta-heuristics are high-level generic processes for generating heuristics to tackle a specific problem. Genetic algorithms, ant colony optimization, particle swarm optimization, and other meta-heuristic approaches are examples of meta-heuristic techniques. When the mapping between the tasks and the resources is generated is determined by the mapping approach. Static and dynamic mapping are the two different mapping procedures employed.

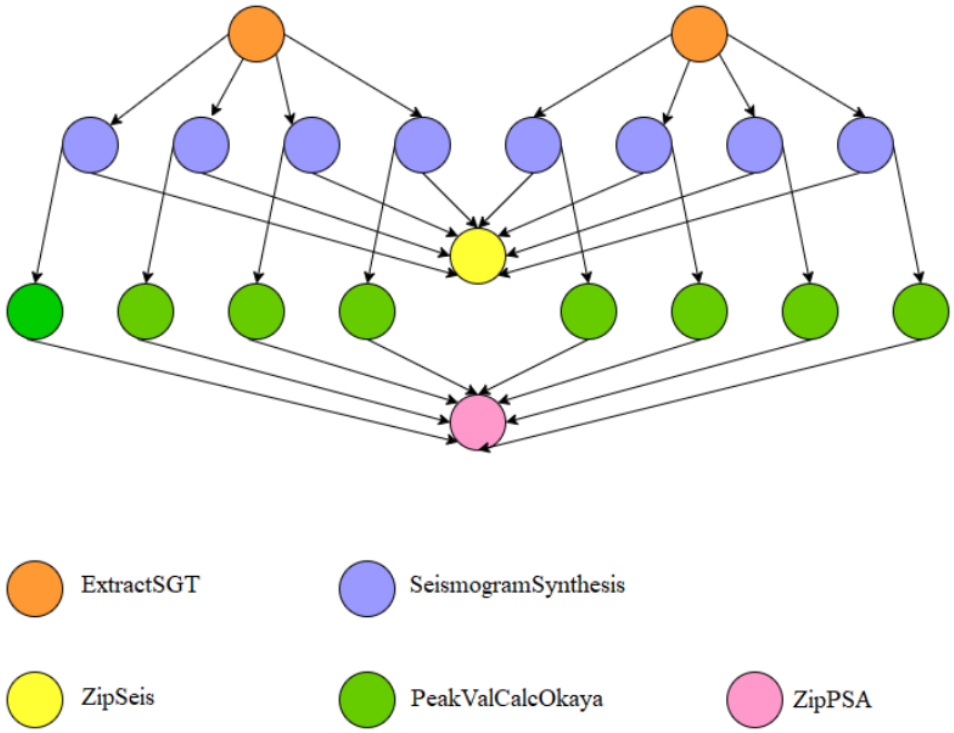

Fig. 1: A sample cyber shake workflow

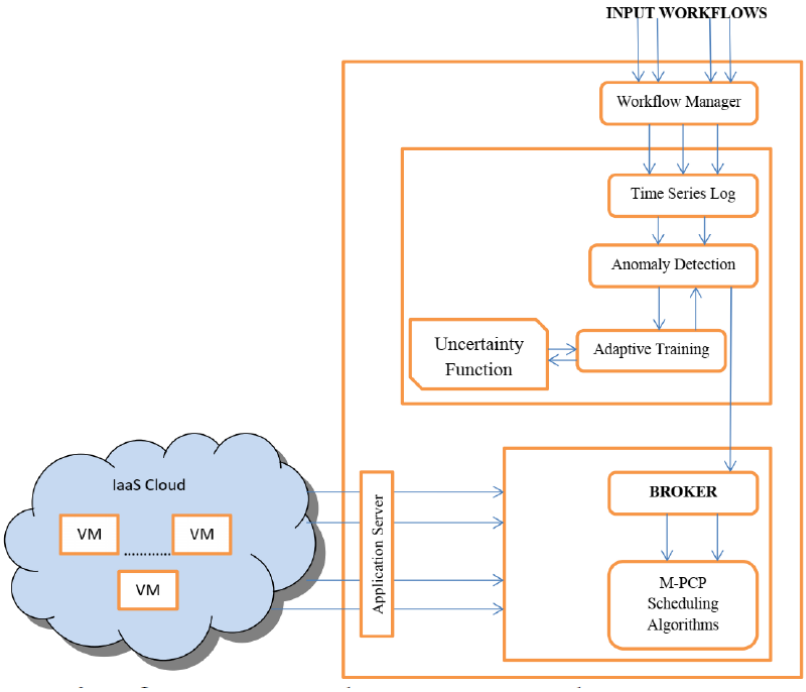

Fig. 2: Proposed system architecture

Static Scheduling

Santra and Mali (2015) introduced a circular approach to the Round Robin concept, intending to clarify the load balancing scenario of a cloud server during execution. It aids in the creation of an effective communication architecture between broker and Virtual Machine (VM) to optimize time and reduce costs by providing an effective and quick execution environment for tasks provided by the user. They use cloud sim 3.0 and VM scheduling (Space and Time sharing) policies to implement it. Round Robin and FCFS scheduling policies are being investigated for Virtual Machine and Cloudlet scheduling. Joshi and Kumari (2016) suggested cloud computing as a type of networked computing that improves its efficiency, accessibility, and utility. From conception to implementation, from consumption to maintenance, cloud implementation is divided into several phases. The efficient usage of cloud computing is dependent on several factors, including security, speed, and privacy. By altering the basic load balancing algorithms, they were able to make better use of cloud resources and boost access speed. The allocation of virtual computers to user bases is also improved and cloud bus' cloud sim toolset is used for verification.

Alam *et al*. (2016) predicted that the number of internet users would continue to rise and that Cluster-Based Web Servers (CBWS) would see a significant increase in online traffic. Because of its simplicity, the Load-Balancing method based on the round-robin Algorithm (RLBA) is most extensively utilized for dispersing loads among web servers. RLBA load distribution, on the other hand, is inefficient in the case of non-uniform web traffic. The proposed unique ways to optimize the RLBA using the essential procedures in this research. They suggest two types of RLBA: Adaptive RLBA (ARLBA) and predictive RLBA (PRLBA). They use simulation data to verify the effectiveness of algorithms. The performance measuring parameters are server load correlation and load variance. In every scenario, ARLBA and PRLBA exceed RLBA. In the case of ARLBA, it outperforms Modified Round Robin (MRR), while in the case of PRLBA, it outperforms MRR.

Dynamic Scheduling

Banyal and Ojasvee (2016) presented a cloud computing system that uses resource management to deliver an elastic, scalable resource-sharing service. In the cloud computing context, resource automation and high-performance management are built on the foundation of resource monitoring as well as prediction. The paper labels the problem of resource watching as well as a prediction in a cloud-computing environment, outlines and executes a flexible resource watching structure in cloud computing, and also proposes one resource forecasting structure contingent on Vector-Auto-Regression (VAR) by the interrelationship among various resources.

Table 1: Scheduling algorithms using VM allocation and parameter-based evaluation

Reference	Scheduling algorithm	VM allocation static/dynamic	Parameter used
Santra and Mali (2015)	Round-robin	Static	Running time as well as resources scheduling
Joshi and Kumari (2016)	Modified RR	Static	Resource allocation and utilization
Alam *et al*. (2016)	Adaptive and predictive round-robin	Static	Performance optimization
Banyal and Ojasvee (2016)	Resource prediction mechanism using VAR	Dynamic	Resource monitoring and prediction
Mehta *et al*. (2017)	Resources monitored using HMM	Dynamic	Execution time height

Experimental evaluations verify the suggestion that resource monitoring structure successfully detects resource utilization in a cloud-computing situation and that the forecasting method dependent on vector-auto-regression is more effective at predicting resource usage than alternative prediction mechanisms.

Mehta *et al*. (2017) propose cloud computing as a concept that utilizes resource management to deliver elastic, scalable, and resource-sharing services. Resource watching and forecasting are the keys to attaining resource utilization with high-performance management in cloud computing. One of the most difficult aspects of cloud computing is resource scheduling; the scheduling strategy and algorithm have a direct impact on the cloud system's performance. Due to resource limits, cloud computing has recently introduced high-performance computing capacity, urging cloud providers to fully utilize resources. The purpose of this study is to employ a Hidden Markov Model to monitor the cloud resources available (HMM). The suggested model is utilized to track resource availability, after which the resource is classed as light, average, or heavily loaded, and the best scheduling strategy is chosen based on demand. The algorithm's efficiency has been tested using the workload scenarios in Table 1.

Proposed Workflow Manager and Time Series Logs

The Workflow Manager receives the input workflows and is in charge of recording information for each action and providing tools for reporting on each operation. To acquire time series logs, each activity in the workflows is evaluated and these log entries have been resolved in the direction of an instantaneous time sequence consisting of three features (three dimensions). The difference in time between adjacent log entries, which is determined with millisecond accuracy is the first dimension. The next dimension compares the differences in the tasks/processes completed by linking two logs. Since each task in the Scientific process is to be performed in a separate physical location, the difference in IP address between two consecutive log entries would be represented by the last dimension. Sub-sequences are designed and submitted to the anomaly detection module as log entries are resolved, which detects anomalies using a modified matrix profile.

System Model

An IaaS cloud model, an application server, and Customer applications for scheduling make up the proposed scheduling system concept Fig. 2. A client application is modeled as a directed acyclic graph $G = (A, E)$, where A is the collection of n activities (tasks) $t_1, t_2, ..., t_n$, and E is the collection of dependencies. Every dependency $e_{i,j} = (t_i, t_j)$ represents a precedence constraint, indicating that activity t_i must complete execution before activity t_j can begin. A task with no parent is called the entrance task, whereas a task without children is called the exit task of the given task graph. We've added fake activities $_{entry}$ as well as t_{exit} as the first and last tasks in the given workflow, respectively because the suggested algorithm needs a unique arrival and leaving task. The computation times of newly added tasks are considered to be zero with zero weight dependency added to or from existing arrival and leaving tasks.

The proposed concept is built on pay for a usage pricing model, analogous to what public Clouds offer. Cloud users are debited for the total amount of periods in which they have consumed the resources for complete intervals as well as for partial intervals. The cost of computing service s_i for a single time interval is assumed to be c_i in this study. ET (t_i,s_j) is also the time it takes to complete task t_i on computational service s_j. All of the services (computational and storage) are meant to be in the same physical location, with nearly similar average bandwidth among the computational services. TT (e_{ij}), data transfer time for dependency $e_{i,j}$, is determined solely by the size of data exchanged between tasks t_i and t_j and is unaffected by the services that perform them. When both tasks t_i and t_j are running on the same computational service, TT (e_{ij}) is equal to zero. Because the cost of data transfer in most real clouds is zero, we assume that the cost of data transfer in our model is similarly zero. In real clouds, clients are charged for storage services based on the size of the assigned storage volume and the quantity of input-output activities to/from the exterior of the cloud. We don't include these parameters in our model because they have no bearing on our algorithm.

Anomaly Detection

A modified matrix profile model is used in the anomaly detection phase. The following are the primary changes made to the Matrix Profile utilized here:

- Instead of computing the Euclidean distance concerning all the existing sub-sequences and determining the lowest distance for a sub-sequence, a semi-supervised model that uses M-m+1

sub-sequences from the start as references to comparison with no anomalies

- Rather than computing the absolute Euclidean distance, calculate the relative distance between sub-sequences

The relative distance between two univariate sub-sequences $T_{1,\,m}$ = [t_1, t_2, …., t_m] and $T'_{1,m}=\left[t'_1,t'_2,....,t'_m\right]$ is computed as:

$$relative\ distance\ rd=\frac{\sum_{l=1}^{m}\left|T_{k,m}^{(j)}[l]-T_{i,m}^{(j)}[l]\right|}{\sum_{l=1}^{m}\left|T_{k,m}^{(j)}[l]\right|} \quad (1)$$

where, k varies from 1 to M-m+1.

During each pace, the Anomaly-Detection approach uses a sub-sequence of the input stream and the uncertainty function provided as input and produces betai, a comprehensive anomaly score that indicates whether or not the given sub-sequence is an abnormality. The comprehensive anomaly score produced by the module is a numeral value in the range of 0 to 1 and when it is close to 1, the sub-sequence is considered anomalous. There will be $(M-m+1)^d$ viable combinations because there are M-m+1 sub-sequences and d dimensions and the algorithm retains a weight for those combinations using W stored in a hash table indexed by a key. By considering the adjacent sub-sequences and removing those tasks in overlapping anomalous sub-sequences.

Anomaly-score (β_i) collects the findings of various Matrix-Profile models operating on different features of a specific sub-sequence. The anomaly detection module can also provide information about the individual aspects that contribute to the anomaly. Our anomaly detection method employs a semi-supervised framework in which it determines the exact execution performance of workflows over a few workflow rounds and recognizes the cases that gradually diverge from this as possible abnormalities. Algorithm 1 describes the anomaly detection algorithm.

Adaptive Training

The system thinks that the anomaly flagged is a true positive with high certainty if the probabilistic value is near 1 and the converse is assumed if the probabilistic value is close to 0. The uncertainty function assumes that the chances of anomalies arising during the first few runs are modest and that the scenario evolves as the attacker Adaptive Training method is described in Algorithm 2. When the method is run, the weights are updated once the anomaly-detection method detects an abnormality. Once an abnormality is identified, the weights are modified considering the uncertainty function and anomaly-score β_i.

Algorithm1: Anomaly detection

/* The Anomaly Detection
algorithm is executed for each
time step *i* */

Input:

i, the time step
$T_{i,m}=\left[T_{i,m}^{(1)},T_{i,m}^{(2)},...,T_{i,m}^{(d)}\right]$, the sequence of input time
m, the length of the sub-sequence
M, the window size
T, sample sub-sequence for comparison
R, *H*, the records to be updated for adaptive training
d, the number of dimensions
θ, the user-defined threshold
W, the weights

Output:

Anomaly detected, a boolean value
C, the contribution list
R, *H*, updated records for adaptive training

1min r^d = 0; **for** each dimension *i* **do**
2 Compute the relative distance using Eq. 1
3 update *min-rd* if needed (if the current rd is less than $_{\min}r^d$)
4 Compute *key*, β_i and D $_{min}[i]$
5 save β_i into the contribution list C_i
6 update R
7 if *key* ∈ *W* **then**
8 *update* β_i as β_i * W[*key*]
9 *update H*
10 if β_i >θ **then**
11 *Anomaly-Detected* = TRUE
12 else
13 *Anomaly-Detected* = FALSE
14 return [*Anomaly-Detected, C_i, R, H*]
15

Algorithm 2: Adaptive training

/*The adaptive training
algorithm is executed for each
time step *i* */

Input:

i, *m*, *M*, *q*, *d*, *W*, *R* and *H*, same as in Algorithm 1
P_i, the uncertainty value at time step *i*
a, the training bias value

Output:

W, the weight
H, Upadated information for adaptive training.

1 for *each k, the weight value selected* **do**
2 *update keys*[*k*]
3 **if** *W does not contain keys*[*k*] **then**
4 update *W* [*keys*[*k*]] as 1
5 if *k* = = *m*+1 **then**
6 update β_i and *H*
7 return [*W*, *H*]

Broker and Modified PCP Scheduling Algorithm

The broker receives the anomaly information for each task in the workflow and modifies the workflow DAGs by removing the anomalous tasks from the workflows, as well as the dependencies associated with the anomalous tasks.

The broker is also in charge of adding two zero-execution-time dummy tasks, t_{entry} and t_{exit}, which are linked to the actual entry and exit tasks with zero weight dependencies. The Scheduling algorithm receives the changed workflows from the Broker. In this case, we're utilizing a modified version of the original PCP algorithm (Abrishami *et al*., 2013), as described in Algorithm 3. The services covered here are diverse and when launching a new service, the VM start-up time is taken into account. The Proposed scheduling module has two ideas for activity start times: The Earliest Start Time (EST), which is calculated before the workflow is scheduled, and the Actual Start Time (AST), which is derived later the tasks are scheduled.

Algorithm 3: Modified PCP

/**Modified PCP* algorithm is executed for the scientific workflow. */

Input:

The modified *DAG G* = (*V*, *E*) from the BROKER. 1

D, the deadline for executing the workflow.

Output:

Cost of executing the Workflow

1 Compute the execution time of each task in each of the available services
2 Compute EST and EFT for all tasks in the DAG for the fastest available service
3 Compute LFT for all tasks in the DAG for the fastest available service
4 Compute the average Data Transfer time between the tasks
5 $t = tex_{it}$
6 **while** t ≠ t_{entry} **do**
7 **while** t has an unassigned parent **does**
8 find the Partial Critical Path (*PCP*) of *t* with only unassigned parents
9 set s_i, *j* as the cheapest instance *j* of available service *si*, which is already allocated to the workflow, which can finish all the tasks the *PCP* before their *LFT*
10 **if** (*si*, j^{is} null) **then**
11 launch a fresh instance of the least cost service that could finish all the tasks in the PCP(t) before their *LFT-boot time* (service)
12 **for** (each task t_i in *PCP* (*t*)) **do**
13 Schedule *t*i on *si*, *j*
14 set *SS* (t_i) and AST (t_i)
15 set assigned (t_i) = *true*

Earliest-Start-Time (EST)

EST (t_i), the earliest start-time of an unscheduled task t_i is defined as:

$$EST\left(t_{entry}\right) = 0 \tag{2}$$

$$EST(t_i) = \max_{t_p \in t_i's\ parents} \left[EST\left(t_p\right) + MET\left(t_p\right)(+TT\left(e_{p,i}\right)\right] \tag{3}$$

where, $MET(t_i)$ is the minimum time to execute a task t_i, which is described as the time to execute task t_i on a service $s_j \varepsilon S$, and $ET(t_i,s_j)$, which is the minimum among the available services. Notice that $MET(_{entry})$, as well as $MET(t_{exit})$, are set to zero. $TT(e_{p,\ i})$ is the time for data transfer over the dependency e_{pi}.

Earliest-Finish-Time (EFT)

For an unscheduled tasks t_i, *EFT* (t_i), is defined as:

$$EFT\left(t_i\right) = EST\left(t_i\right) + MET\left(t_i\right) \tag{4}$$

Latest Finish-Time (LFT)

Latest Finish-Time of an unscheduled task t_i, $LFT(t_i)$ is defined as the latest time where t_i can finish its execution so that the entire workflow can complete its execution before the user-specified deadline, *D*. It is calculated as:

$$LFT\left(texit\right) = D \tag{5}$$

$$LFT\left(t_i\right) = \min_{tc \in t_i's children} \left[LFT(t_C) - MET\left(t_C\right) - TT(e_{i,c}) \right] \tag{6}$$

Selected Service (SS): The Selected Service of every scheduled task t_i, $SS(t_i) = s_{j,k}$, is the service chosen for executing t_i during scheduling, and $s_{j,k}$ is the k^{th} instance of service s_j.

Actual Start Time (AST)

The actual start time of any task t_i, *AST* (t_i), is termed as the time in which the task t_i starts on the selected service and this is calculated only after scheduling.

Assigned-Node

An assigned node is the node that is already scheduled to service and indicates that the selected service of task is identified.

Critical Parent

t_i's parent, which is not yet assigned to any resource and has the last data reaching time at t_i. i.e., it's t_i's parent t_p whose $EFT(t_p)+TT(e_{p,i})$ is maximum.

Partial Critical-Path (PCP)

For every node t_i, (*PCP*) is the most important concept in the proposed scheduling algorithm, which is defined as:

- *PCP* of t_i is empty when t_i do not possess unassigned parents
- if t_i has an unassigned parent t_p, then $PCP(t_i)$ includes t_p and $PCP(t_p)$

The Partial Critical Path (PCP) of a task *t* is computed as–consider all the unassigned parents of *t* and out these unassigned parents, the parent t_p whose *EST* + *MET* + *TT* is maximum belongs to *PCP*(*t*) and continue adding parent tasks of t_p to *PCP*(*t*) until there are no unassigned parents left, or we reach t_{entry}.

Once the PCP of a task is obtained, the scheduler checks, whether, there is an existing instance, which is cheaper and which can finish all the tasks in the PCP before its LFT. It is not available, it launches a new service instance and takes care of the boot-up time of the new service and schedules all the tasks in the PCP of *t* to the same service, and updates the Selected Service (SS) and Actual Start Time (AST) of every task t_i in PCP (*t*).

Performance Analysis

The proposed scheduling algorithm is evaluated on two Scientific Workflow–Montage and Cyber shake. Developed a workflow generator, which creates workflows of an arbitrary size similar to the real-world Scientific Workflow. These workflows are available in DAX (Directed acyclic graph in XML) format on their website (PWMS), from which we choose three sizes for our experiments–small, large, and extra-large Table 2. The cloud computing Environment is simulated using CloudSim (Calheiros *et al*., 2011).

The input workflows are first analyzed for anomalies. Experimental results show that an average of 15% of tasks in the workflows are anomalous tasks. The anomalous tasks are eliminated from the workflows and the resulted workflows are scheduled using a modified PCP algorithm. Our proposed algorithm is compared with the IC-LOSS (LOSS algorithm for IaaS clouds) algorithm, a modified version of the original LOSS algorithm used with Grids (Sakellariou *et al*., 2007) and ICPCP algorithm (Abrishami *et al*., 2013). For our experiments, we assume an IaaS Cloud environment with 7 different computational services (same as in Amazon EC 2) with varying prices and performances Table 3.

The bandwidth between computational services is set to 20 Mbps on average (same as in EC 2). Amazon charges users for the large time interval of an hour, but cloud sigma prefers a shorter time of 5 min. In our trials, we used two distinct time intervals of one hour and five min, respectively. Because we are evaluating a large number of workflows with various sorts of attributes in our studies, we have defined a parameter called Normalized cost (which is comparable to the Normalized Deadline (ND) used in Anwar and Deng, 2018):

$$Normalized\ Cost = \frac{Total\ Scheduled\ Cost}{Cheapest\ Cost} \tag{7}$$

where, *Cheapest Cost* is defined as the total cost to execute the same workflow in the cheapest service available.

To assign a deadline to the input workflows, we are defining the term Fastest Executing Time (F_T), as the time obtained when each task in the work is allocated to the fastest service available and considering the data transmission time as zero, which is a theoretical measure.

The deadline D is computed as:

$$D = \alpha . F_T \tag{8}$$

where, *a* varies from 1.5 to 5 with a step value of 0.5.

Figure 3 shows the workflow scheduling cost for the large workflows in IC-PCP, IC-LOSS and M-PCP for the time interval of 1 hour which are tabulated in Table 4 and Table 5 and Fig. 4 shows the workflow scheduling cost for the same workflows for the time interval of 5 min which are tabulated in Table 6 and 7. It is clear that all the methods successfully scheduled incoming workflows over their allotted deadlines, even for tight deadlines where D = 1.5* F_T *or D* = 2* F_T.

Table 2: The applications used and their sizes

Application	Small	Large	Extra large
Cyber shake	30 nodes	100 nodes	1000 nodes
Montage	25 nodes	100 nodes	1000 nodes

Table 3: VM types used

VM Type	vCPU	Memory	Price
a 1. Medium	1	2 GB	$ 0.0255 per h
m 6 g. Medium	1	4 GB	$ 0.0385 per h
t 3. Small	2	2GB	$ 0.0208 per h
t 3. Medium	2	4 GB	$ 0.0416 per h
t 3. Large	2	8 GB	$ 0.0832 per h
t 3a. Large	4	16 GB	$ 0.1504 per h
t 2. 2 × Large	8	32 GB	$ 0.3712 per h

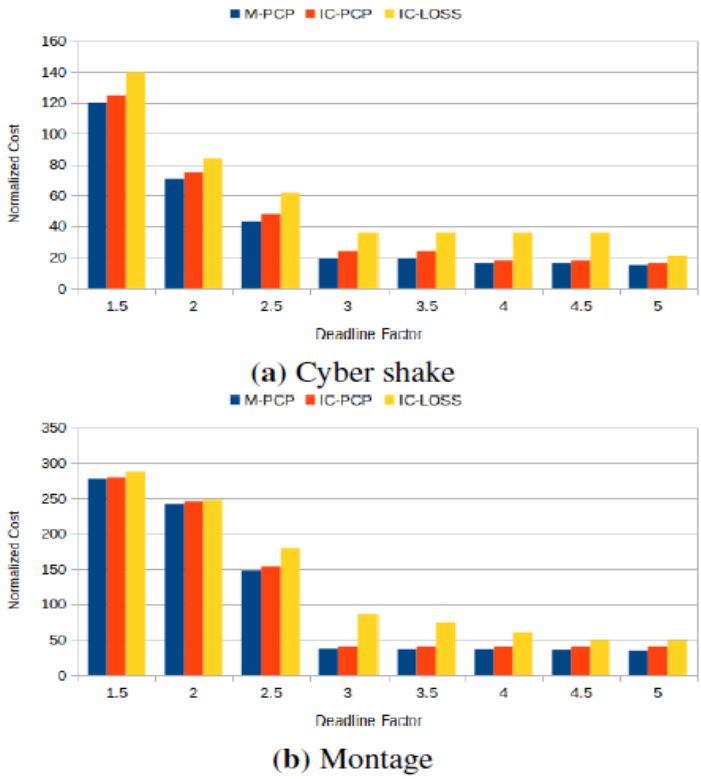

Fig. 3: The cost of scheduling workflows with the time interval of 1 h

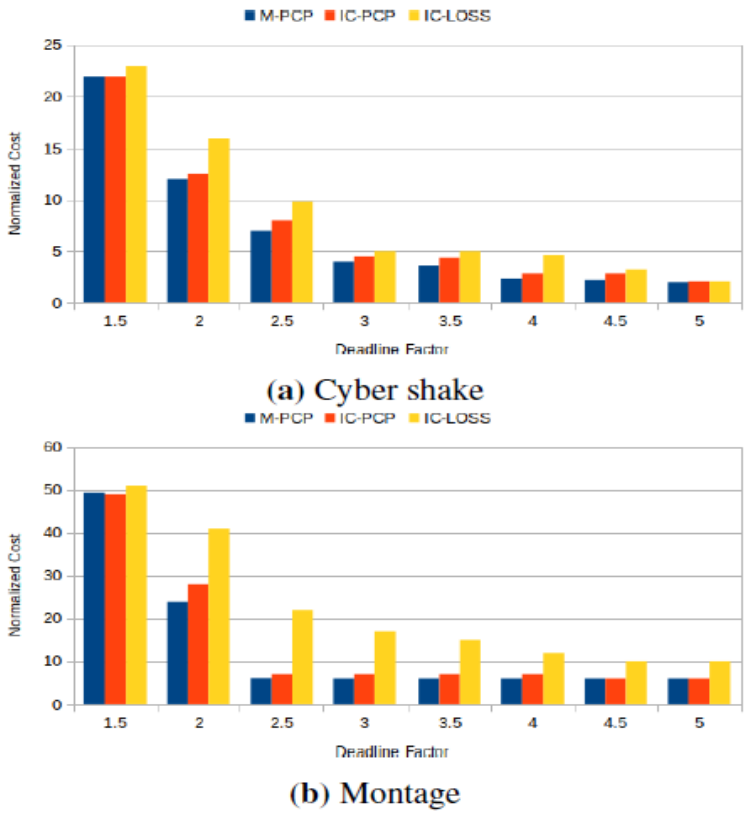

Fig. 4: The workflow scheduling cost for the time interval of 5 min

Table 4: Performance of Cyber shake workflow for billing interval 1 h

α	M-PCP	IC-PCP	IC-LOSS
1.5	119	125	140
2.0	71	75	84
2.5	43	48	62
3.0	19	24	36
3.5	19	24	36
4.0	16	18	36
4.5	16	18	36
5.0	15	16	21

Table 5: Performance of montage workflow for billing interval 1 h

α	M-PCP	IC-PCP	IC-LOSS
1.5	278	280	288
2.0	242	246	248
2.5	148	154	180
3.0	37	40	86
3.5	36	40	74
4.0	36	40	60
4.5	36	40	50
5.0	36	40	49

Table 6: Performance of Cyber shake Workflow for billing interval 5 min

α	M-PCP	IC-PCP	IC-LOSS
1.5	22.00	22.0	23.0
2.0	12.00	12.5	16.0
2.5	7.50	8.0	9.8
3.0	4.25	4.5	5.0
3.5	4.00	4.4	5.0
4.0	2.30	2.8	4.6
4.5	2.20	2.8	3.2
5.0	2.00	2.1	2.1

Table 7: Performance of Montage Workflow for billing interval 5 min

α	M-PCP	IC-PCP	IC-LOSS
1.5	49.5	49	51
2.0	28.0	28	41
2.5	6.9	7	22
3.0	6.8	7	17
3.5	6.7	7	15
4.0	6.7	7	12
4.5	6.0	6	10
5.0	6.0	6	10

All the results show that M-PCP beats IC-PCP as well as IC-LOSS in nearly all situations, even if the boot-up time for the newly launched instances is considered. From the results, we can see that the Normalized Cost is very high when the Deadline factor α is 1.5, the reason for the high values is that, as the deadline becomes very tight, it is required to allocate a new instance to each new task to finish the workflow before its specified deadline and many of the tasks are using a small fraction of the allocated instances. The results in Fig. 3 and 4 show that we can produce cost savings in scheduling when the deadline factor $\alpha \geq 2.5$. When the time interval is short (5 min), we get high-cost savings on all deadlines, and *M-PCP* outperforms.

Conclusion

From experiments, it is seen that some of the tasks in the workflows are anomalous tasks. If the anomalous tasks are also considered for scheduling in a cloud environment, it will increase the cost of cloud usage. In this study, we are considering scheduling of workflows as a two-step process-first step is a preprocessing step, which detects anomalous tasks in the workflows and the experiments show that an average of 15% of the tasks are anomalous tasks, and resulting workflows without anomalous tasks are scheduled in heterogeneous cloud environment using a modified PCP algorithm. Here the modification used is, that the cloud environment is more close to the real cloud environment, where the cloud boot-up time is also considered. The experiments are conducted on two scientific workflows-Cyber shake and Montage. The proposed algorithm is compared with IC-PCP and IC-LOSS algorithms for different deadlines and analyzed the cost of cloud usage and the results show that M-PCP outperforms IC-PCP and IC-LOSS algorithms. In the future, we will extend our work to identify the nature of anomalies in the workflow tasks, if the task is anomalous.

Acknowledgment

The research was done as part of my (first author) Ph. D. work, with second and third authors as my Internal Supervisors and fourth author as my external supervisor. All the experiments are conducted in Karpagam Academy of Higher Education, Coimbatore, Tamilnadu, India and this research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Author's Contributions

Anver Shahabdeen Rahumath: Investigation, problem formulation, methodology, formal analysis, software implementation, data analysis, original draft paper preparation.

Santhosh Rajendran, Natarajan Mohanasundaram and Abdul Rahiman Malangai: Supervision, design research plan, research administration, problem formulation, methodology, resources, writing review, draft paper correction and editing.

Ethics

This article is an original research work. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Abrishami, S., Naghibzadeh, M., & Epema, D. H. (2013). Deadline-constrained workflow scheduling algorithms for infrastructure as a service cloud. Future generation computer systems, 29(1), 158-169. doi.org/10.1016/j.future.2012.05.004

Alam, F., Thayananthan, V., & Katib, I. (2016, August). Analysis of round-robin load-balancing algorithm with adaptive and predictive approaches. In 2016 UKACC 11th international conference on control (pp. 1-7). IEEE. doi.org/10.1109/CONTROL.2016.7737592

Alguliyev, R. M., Aliguliyev, R. M., & Abdullayeva, F. J. (2019). The hybridization of classifiers for anomaly detection in big data. International Journal of Big Data Intelligence, 6(1), 11-19. https://www.inderscienceonline.com/doi/abs/10.1504/IJBDI.2019.097396

Anshu, B., Kaveri, P. R., Singh, H., & Chavan, V., (2020). Cloud resource management: Comparative analysis and research issues. International Journal of Scientific and Technology Research, 9(6), 162- 169. https://www.researchgate.net/publication/341701698_Cloud_Resource_Management_Comparative_Analysis_and_Research_Issues

Anwar, N., & Deng, H. (2018). Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments. Future Internet, 10(1), 5. doi.org/10.3390/fi10010005

Banyal, R. K., & Ojasvee, K. (2016). Anal. and improve. Of load balancing in cloud computing. (ICTBIG) Int. Conf. on ICT in business industry government, pp. 1-5. doi.org/10.1109/ICTBIG.2016.7892711

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience, 41(1), 23-50. doi.org/10.1002/spe.995

Dabbagh, M., Hamdaoui, B., Guizani, M., & Rayes, A. (2015). Toward energy-efficient cloud computing: Prediction, consolidation and overcommitment. IEEE Network, 29(2), 56-61. doi.org/10.1109/MNET.2015.7064904

Chen, D., Chen, L., Shao, G., Li, H., Yin, X., & Tao, S. (2018). Research of security as a service for VMS in the IaaS platform. IEEE Access, 629158-29172.

Rynge, M., Poehlman, W., Feltus, F., Vahi, K., Deelman, E., Mandal, A., Baldin, I., Bhide, O., Heiland, R., Welch, V. & Hill, R., (2019). Integrity protection for scientific workflow data: Motivation and initial experiences. pages 1-8, 07 2019. ISBN 978-1-4503-7227-5. doi.org/10.1145/3332186. 3332222

Genez, T. A., Bittencourt, L. F., & Madeira, E. R. (2020). Time-discretization for speeding-up scheduling of deadline-constrained workflows in clouds. Future Generation Computer Systems, 107, 1116-1129. doi.org/10.1016/j.future.2017.07.061

Hu, Z., Gnatyuk, S., Koval, O., Gnatyuk, V., & Bondarovets, S. (2017). An anomaly detection system in the secure cloud computing environment. International Journal of Computer Network and Information Security, 9(4), 10. doi.org/10.5815/ijcnis.2017.04.02

Snir, M., Wisniewski, R., Abraham, J., Adve, S., Bagchi, S., Balaji, P., Belak, J., Bose, P., Cappello, F., Carlson, B., Chien, A., Coteus, P., DeBardeleben, N., Diniz, P., Engelmann, C., Erez, M., Fazzari, S., Geist, A., Gupta, R., Johnson, F., Krishnamoorthy, S., Leyffer, S., Liberty, D., Mitra, S., Munson, T., Schreiber, R., Stearley, J., & Hensbergen, E. V. (2014). Addressing failures in exascale computing. International Journal of High Performance Computing Applications. ISSN 1094-3420. doi.org/10.1177/1094342014522573.

Torre, E., Durillo, J. J., Maio, V. D., Agrawal, P., Benedict, S., Saurabh, N., & Prodan, R. (2020). A dynamic evolutionary multiobjective virtual machine placement heuristic for cloud data centers. Information and Software Technology, 128, 2020. doi.org/10.1016/j.infsof.2020.106390

Körber, N., Geldreich, K., Stahlbauer, A., & Fraser, G. (2021, May). Finding anomalies in scratch assignments. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) (pp. 171-182). IEEE. doi.org/10.1109/ICSE-SEET52601.2021.00027

Mehraj, S., & Banday, M. T. (2021). A flexible fine-grained dynamic access control approach for the cloud computing environment. Cluster computing, 24(2), 1413-1434. doi.org/10.1007/978-3-642-03829-7_5

Mehta, H., Prasad, V. K., & Bhavsar, M. (2017). Efficient resource scheduling in cloud computing. International Journal of Advanced Research in Computer Science, 8(3), 809-815.

Osypanka, P., & Nawrocki, P. (2020). Resource usage cost optimization in cloud computing using machine learning. IEEE transactions on cloud computing. doi.org/10.1109/TCC.2020.3015769

Pan, Y., Sun, X., Xia, Y., Zheng, W., & Luo, X. (2020, November). A Predictive-Trend-Aware and Critical-Path-Estimation-Based Method for Workflow Scheduling Upon Cloud Services. In 2020 IEEE International Conference on Services Computing (SCC) (pp. 162-169). IEEE. doi.org/10.1109/SCC49832.2020.00029

Rodriguez, M. A., & Buyya, R. (2018). Scheduling dynamic workloads in a multi-tenant scientific workflow as a service platform. Future Generation Computer Systems, 79, 739-750. doi.org/10.1016/j.future.2017.05.009

Rodriguez, M. A., Kotagiri, R., & Buyya, R. (2018). Detecting performance anomalies in scientific workflows using hierarchical temporal memory. Future Generation Computer Systems, 88, 624-635. doi.org/10.1016/j.future.2018.05.014

Gaikwad, P. Mandal, A., Juve, G., Krol, D., & Deelman. E. (2016) Anomaly detection for scientific workflow applications on networked clouds. Inter. Conf. on High-Performance Computing. doi.org/10.1109/HPCSim.2016.7568396

Sakellariou, R., Zhao, H., Tsiakkouri, E., & Dikaiakos, M. D. (2007). Scheduling workflows with budget constraints. In Integrated research in GRID computing (pp. 189-202). Springer, Boston, MA. doi.org/10.1007/978-0-387-47658-2_14

Santra, S., & Mali, K. (2015, September). A new approach to survey on load balancing in VM in cloud computing: Using cloud sim. In 2015 International Conference on Computer, Communication and Control (IC4) (pp. 1-5). IEEE. doi.org/10.1109/IC4.2015.7375671

Somani, G., Gaur, M. S., Sanghi, D., Conti, M., & Buyya, R. (2017). DDoS attacks in cloud computing: Issues, taxonomy and future directions. Computer Communications, 107, 30-48. doi.org/10.1016/j.comcom.2017.03.010

Sridharan, R., & Domnic, S. (2021). Network policy-aware placement of tasks for elastic applications in IaaS-cloud environment. Cluster Computing, 24(2), 1381-1396. doi.org/10.1007/s10586-020-03194-z

Tiwari, A., & Garg, R. (2021). ACCOS: A Hybrid Anomaly-Aware cloud computing formulation-based ontology services in clouds. http://ceur-ws.org/Vol-2786/Paper42.pdf

Joshi, S., & Kumari, U. (2016, December). Load balancing in cloud computing: Challenges and issues. In 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I) (pp. 120-125). IEEE. doi.org/10.1109/IC3I.2016.7917945

van Zelst, S. J., van Dongen, B. F., van der Aalst, W. M., & Verbeek, H. M. W. (2018). Discovering workflow nets using integer linear programming. Computing, 100(5), 529-556. doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

Varshney, P., & Simmhan, Y. (2020). Characterizing application scheduling on edge, fog and cloud computing resources. Software: Practice and Experience, 50(5), 558-595. doi.org/10.1002/spe.2699