JSR Publications

Available Online

J. Sci. Res. **13** (2), 423-438 (2021)

JOURNAL OF SCIENTIFIC RESEARCH

www.banglajol.info/index.php/JSR

A Task Scheduling Approach for Cloud Environments Employing Evolutionary Algorithms

B. Lakhani*, A. Agrawal

Department of CSE, Medicaps University, Indore, (M.P.), India

Received 25 October 2020, accepted in final revised form 12 February 2021

Abstract

One of the key challenges in the domain of cloud computing is task scheduling and estimation of cloud workloads for time critical applications pertaining to constrained cloud resources. While effective task scheduling is necessary for balancing the load, workload forecasting is necessary to plan in advance the requirements of cloud platforms based on previous data so as to effectively utilize cloud resources. Often it is challenging to gather sufficient information about the tasks and hence allocating the tasks to virtual machines (VMs) in the most optimal way is non-trivial. In this paper, a hybrid task scheduling approach is proposed based on evolutionary algorithms. The first approach is the amalgamation of bat and particle swarm optimization (PSO) techniques. The scheduling approach also combines the processing time preemption (PTP) approach to schedule the source intensive tasks which allows to reduce the response time of the proposed system. The second approach is a machine learning based approach employing gradient descent with momentum (GDM). The evaluation of the proposed system has been done based on the response time and mean square error of the system.

Keywords: Cloud Computing; Task Scheduling; Particle Swarm Optimization; Bat Optimization; Gradient descent with momentum (GDM); Processing time preemption (PET).

doi: http://dx.doi.org/10.3329/jsr.v13i2.49944 J. Sci. Res. **13** (2), 423-438 (2021)

1. Introduction

Cloud computing has seen tremendous increase in terms of applications which need sophisticated computation or memory applications. Typically, such applications need on-demand memory and/or computational resources. Virtual machines (VMs) are used to emulate a computer and are typically deployed on a host computer which seeks cloud based services. It is possible based on virtualization to create multiple VMs on a single physical machine with each VM having its own operating system (OS). Typically, a software termed as the hypervisor is used as a coordination link between the VM and the physical hardware. Cloud resources are constrained and hence it is mandatory to design algorithms which can render swift provisioning and release of the shared resources. This in turn makes it critical for the system to possess a low response time to avoid latencies.

* *Corresponding author*: bhawanalakhani27@gmail.com

To allocate and balance the requirements of clients, effective task scheduling is needed which fulfils the criteria of overall balancing of the system as well as maintaining least amounts of latency. Without loss of generality, it can be pointed out that due to the lack of substantial information about the tasks and dynamic allocation of resources, task scheduling is often a heuristic and multi-variate optimization problem. The planning for an upscale or downscale of resources is also required for load balancing which takes us to the other aspect of cloud resource management which is cloud workload forecasting. Since the data to be analysed is typically large for cloud computing applications, hence evolutionary algorithms have been employed off late for task scheduling purposes. In this paper, load balancing is executed employing a hybrid bat-particle swarm optimization (PSO) based approach. The response time has been computed for different tasks as the evaluation parameter for the proposed bat-PSO approach. The scheduling of tasks is done for a particular application that is to be scheduled on pre-created VMs.

2. Related Work

An energy-aware task scheduling has been proposed in literature [1]. The approach proposed an energy-aware run time scheduler to track and minimize the energy consumption of the scheduling process. Adaptive dispatch of tasks for the purpose of load balancing was proposed with Quality of Service (QoS) awareness for the cloud environment [2]. A forecasting model for host CPU utilization was proposed using neural networks by Duggan *et al*. [3]. The approach showed that the forecasting task can be modelled as a time series prediction problem and can be accomplished by using recurrent neural networks. The concept of deep learning was used for resource allocation and management of power [4]. Zuo proposed multi-queue based scheduling based on task classification [5]. The approach showed that classification and clustering of tasks prior to scheduling outperforms conventional heuristic approaches for the same purpose. The concept of a time series model for cloud data centres and cloud workloads was explored [6,7]. The approach showed that it is challenging to find trends in the complex data sets pertaining to cloud computing due to the lack of correlation and repetitive patterns in the data. This can be done employing machine learning algorithms which can optimize a multiobjective function. The most effective techniques can be gradient descent and back propagation [10,11]. It has been shown that the gradient descent with momentum outperforms the conventional gradient descent in terms of the accuracy and number of execution iterations [12]. Thus, the empirical challenges remain to be the effective scheduling of tasks in conjugation with models which could forecast future workload with high accuracy [13,14]. The proposed work tries to design a scheduling algorithm based on the bat-pso algorithms and envisages to attain lesser response time compared to existing heuristic techniques [16]. Additionally, the proposed approach also aims at designing a workload forecasting model for cloud workloads which would attain higher accuracy of workload forecasting in comparison with existing approaches [15,16]. The following

section explains the task scheduling model for the proposed system with an objective to reduce the response time and hence the latency of the system [17-20].

3. The Task Scheduling Model

The modeling of the proposed system is done based on the allotment of the tasks in a workflow to the different resources in a manner so as to minimize the make span and the response time. The first approach for task scheduling model is an amalgamation of the bat and PSO optimization techniques aiming at improving upon the performance of the pre-existent PSO or bat optimization used individually. The second approach is the machine learning based approach with gradient descent with momentum (GDM).

A. *The Bat-PSO Based Approach*

The task scheduling is often an optimization problem with multiple solutions but the one which would optimize the cost function needs to be selected. The proposed approach employs a hybrid bat-PSO approach for optimizing the cost function [8,9]. The particle swarm optimization tries to mimic the movement of a flock of birds. The flock of birds represents the group of available solutions while individual birds represent a particular solution to the optimization problem. In each iteration, each of the solution checks its individual best and the group's best and then decides and updates its best solution. Thus, the PSO is an iterative approach, mathematically governed by [8]:

$$vel_k = w \times vel_{k-1} + L_1\alpha_1[p_i\text{-}x_{i(k-1)}] + L_2\alpha_2[p_g\text{-}x_{i(k-1)}] \quad (1)$$

$$x_{i(k)} = x_{i(k-1)} + vel_k \quad (2)$$

Here,
vel is the particle velocity
k is the iteration
L_1 and L_2 are learning factor values
x_i is the particle position
α_1 and α_2 are random number values
w represents the weights
p_i represents particle's individual best position
p_g represents group's best position

The following is the pseudocode for initializing the PSO algorithm for task scheduling:

```
//number of VMs
int m;
//number of cloudlets
int n;
int numberOfParticles = 10;
int numberOfIterations = 100;
Particle[] swarm;
```

```
    static Random ran = null;
    public int[] getScheduledCloudLets(List<? extends Cloudlet> cloudletList,
List<? extends Vm> vmList) {
```

The Bat algorithm tries to mimic the behavior of bats based on their behavior of communicating and hunting [11,12]. Bat optimization suits workflow scheduling for cloud applications since such problems are NP hard (non-deterministic polynomial time hardness). It is predominantly a meta-heuristic approach for optimization of problems which are multi-variate in nature with multiple possible solutions typically resembling a task scheduling problem. The choice of the solution is made based on the best fit or optimization of parameters to minimize the cost function. The cost function in this case is the response time of the system which decides the delay of the system.
The scheduling algorithm is explained below:
Step. 1: Define the tasks T_i belonging to the resource *S*, mathematically given by:

$$T_1, T_2 \ldots\ldots T_n \in S \quad (3)$$

Step 2: Bat population is initialized with n=1 and cost function (CF) is computed based on the recursive computation costs of tasks '*T*' as:

$$CF[T] = CC[T_k, T_{k+1}] \quad (4)$$

Here, *k* represents the task index.
Step 3: Compute the Transmission Cost (CT) of tasks $T_k \; to \; T_{k+1}$ considering all resources expended in the Set *S*.
Here, the Transmission Cost (CT) is the cost of data transmission between two dependent resources and execution cost of tasks on different resources.
Step 4: The total cost is computed as:

$$TC = \sum_{i=1}^{n} CC_i * CT_i \quad (5)$$

Here, CC is the computational cost and CT is the transmission cost.
The following is the pseudocode to compute the execution time of the cloudlets:

```
//will calculate the execution time each cloudlet takes if it runs on one of the VMs
            for (int i = 0; i < m; i++) {
                Vm vm = vmlist.get(i);
                double[] arr = new double[n];
                for (int j = 0; j < n; j++) {
            Cloudlet cloudlet = cloudletList.get(j);
                        arr[j] = (double) cloudlet.getCloudletLength() /
```

Step 5: The bats are rendered random velocities and corresponding to the different possible solutions based on the following relations:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \quad (6)$$

and

$$v_i^k = v_i^{k-1} + (x_i^{k-1} - x_g)f_i \quad (7)$$

Here, *k* represents the present iteration, *k*-1 represents the past iteration, *g* represents the global best, *f* represents the frequency of communication between the bats, *v* represents the velocity of the bats and β represents the random vector $\in$ [0,1].

The following is the pseudocode to implement the bat optimization:

```
{
      swarm.append(Particle())
      return ( "Swarm initialized.")
   return swarm
public instantantiate_map(obstacles)
{
   new_map = []
   for (i=0;i<=100;i++)
{
If(i==1)
         new_map.append(1)
      else
{
         new_map.append(0)
   return new_map
}
}
}
}
```

Step 6. Find the best solution X such that:

$$\text{Cost} = \min[\text{CT}]\nabla x_i \in X \quad (8)$$

Here, min represents the minimum function, ∇ represents for all values of x_i is the particle position, *X* is the set of all possible particles.

The value of *X* is optimized using the PSO based on equations (1) and (2) to minimize the response time '*T*' for the system and compute response time '*T*'. While using one technique individually such as the bat or the PSO can render its individual best, the amalgamation of two such approaches helps in minimizing the cost function further with respect to what can be achieved using only one technique. This is validated in the comparative results.

Thus the proposed approach is a two-tier method for minimizing the response time and hence the latency.

B. *Processing Time Preemption*

The processing time pre-emption (PTP) based approach tires to rank the tasks depending upon the conditions of existing workload, estimated workload and available bandwidth of the VMs [13,14]. This approach finds out the present condition of the VMs pertaining to running of resource intensive task on the VMs. Prior to pre-empting a task, the condition of the VM is checked to ensure whether its condition is free or occupied. The pre-emptive approach is mathematically modeled as:

Let E(*t*) denoted the time to finish the queued tasks for a VM in a state space of '*T*'. Thus,

$$E(t) = \underbrace{\min\{(\frac{1}{k_m+k_n} + \frac{k_n}{k_m+k_n}}_{} + E*(n) + \frac{k_n}{k_m+k_n} * E(m)\} \quad (9)$$

Here, *t* is the state space of time given by:

$$t\epsilon[t_1 \ldots . t_n] \quad (10)$$

k_m is the time required to execute the task 'm' and k_n is the time required to execute the task 'n'

Let P(n) denote the probability of m being the first task to be completed with a queue of 'n' tasks, then P(n) can be computed as:

$$P(n) = \frac{1}{k_m + k_n} + \frac{k_n}{k_m + k_n} * \mathrm{K}(t_n) \quad (11)$$

Here, the conditional probability of the event T_m is to be computed if it is assumed that n tasks are in the queue based on Baye's theorem of conditional probability given by:

$$P\left(\frac{M}{N}\right) = \frac{P\left(\frac{N}{M}\right).P(M)}{P(N)} \quad (12)$$

Here, M and N are two random events, P(M) is the probability of occurrence or completion of task M, P(N) is the probability of occurrence or completion of task N, $P\left(\frac{M}{N}\right)$ is the probability of occurrence of event M if event N has already occurred, $P\left(\frac{N}{M}\right)$ is the probability of occurrence of event M if event N has already occurred.

Let $L_{m,n}$ denote the latency between the initializing of task 'm' and the completion of task 'n', then the latency can be computed in each iteration of the evolutionary algorithm as:

$$L_{m,n} = T_i^m - T_i^n \quad (13)$$

Here, T_i^m denotes the starting time of task 'm' and T_i^n denotes the ending time of queued tasks 'n'

The aim of the PET approach is to minimize the value of $L_{m,n}$ for the entire span of the iterations of the evolutionary algorithm so as to attain least value of response time. The approach is explained in Fig. 1.

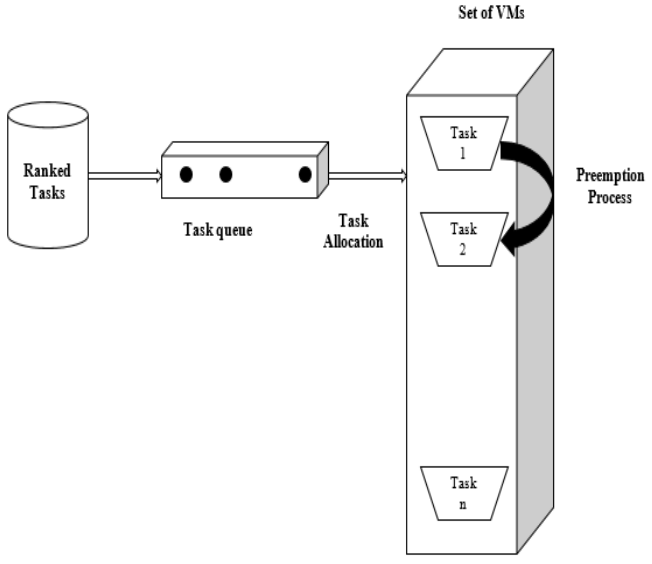

Fig. 1. The Processing Time Preemption (PTP) architecture.

C. *The PSO-GDM Based Approach*

While various machine learning algorithms are popular for optimizing a cost function, one of the most effective choices is the back-propagation algorithm. The back propagation approach is based on the recursive feedback of errors of each iteration to the machine learning model so as to treat is as one of the input vectors thereby enabling the machine learning model to learn from iterative errors [10]. This is one of the most effective ways to train a machine learning model to minimize the objective function [11].

The GDM is a modified version of the gradient descent based approach in machine learning where the cost function is optimized employing a modified and faster version of the gradient descent approach [12]. In the PSO-GDM based approach, the weights of the PSO entities are governed by the GDM algorithm. The essence of this approach is the updating of the gradient vector g, in such a way that it reduces the errors with respect to weights in the fastest manner. The gradient is basically the rate of change of error with respect to weights. Typically in every regression learning algorithm, the aim is to reduce the errors as quickly as possible. This can be accomplished by increasing the value of the gradient 'g'.

Mathematically, let the gradient be represented by g and the descent search vector by p, then

$$p_0 = -g_0 \tag{14}$$

Where, g_0 denotes the gradient given by $\frac{\partial e}{\partial w}$. The sub-script 0 represents the starting iteration and the negative sign indicates a reduction in the errors w.r.t. weights.

The trade-off between the speed and accuracy is clearly given by the following relations:

$$w_{k+1} = w_k - \alpha g_x , \quad \alpha = \frac{1}{\mu} \tag{15}$$

Here, w_{k+1} is the weight of the next iteration, w_k is the weight of the present iteration, g_x is the gradient vector.

Since the error or cost function can be both positive as well as negative, hence squared error or mean squared errors are considered for minimization. The GD tries to minimize the cost function mathematically as [17,18]:

$$minimize < w, \theta > \frac{1}{n}\sum_{i=1}^{n}[y_i - p_i]^2 \tag{16}$$

Here, w represents the weight, θ represents the bias, y is the target or actual output, p is the predicted output and n is the number of samples over prediction.

There are some inherent limitations of the gradient descent approach which are:

1) Oscillations of the gradients
2) Reduction in rate of convergence
3) Higher overshoots

Typically, the response time needs to be minimized by assigning the mean square error of the learning algorithm. The mean square error is defined as:

$$mse = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \tag{17}$$

Here, *mse* stands for the mean square error, n is the number of errors in the iterative learning process, e is the value of the error.

The mean square error is computed as the metric to evaluate the error performance since computing the mean of the squared errors rules out the chance of cancellation of

negative and positive errors in the sum or simple mean computation. Typically it is envisaged to reduce the mse in the least number of iterations.

The cost function is chosen to be the response time of the system [13]. Considering that the error of the system decreases with the number of iterations of training, the acceleration of the error gradient can be mathematically defined as:

$$\vec{a} = \overrightarrow{a_x} + \overrightarrow{a_y} \quad (18)$$

Here, $\vec{a}$ is the net acceleration towards the maximum gradient, $\overrightarrow{a_x}$ is the acceleration of the gradient in the x-direction and $\overrightarrow{a_y}$ is the acceleration of the gradient in the y-direction.

Thus to reach the maximum gradient, the oscillations need to be reduced. This can be done by reducing $\overrightarrow{a_y}$ and somehow increasing $\overrightarrow{a_x}$

Thus the decrease in vertical learning would end up as increase the horizontal learning which is equivalent to damping the vertical acceleration. The gradient descent with momentum tries to essentially accomplish this. Thus for GDM, in each iteration '*k*', $\partial w \; and \; \partial b$ are computed as:

$$v\partial w = \beta v\partial w + (1-\beta)\partial w \quad (19)$$

Similarly,

$$v\partial b = \beta v\partial b + (1-\beta)\partial b \quad (20)$$

And

$$v\partial\theta = \beta v\partial\theta + (1-\beta)\partial\theta \quad (21)$$

The weight is updated as:

$$w_{k+1} = w_k - \alpha v\partial w \quad (22)$$

and the bias is updated as:

$$b_{k+1} = b_k - \alpha v\partial w \quad (23)$$

Here, *k* represents the iteration number, *w* represents the weight, w_k represents the weight of the present iteration, w_{k+1} represents the weight of the next iteration, α is called the learning rate, β is called the friction parameter and $(1-\beta)$ is called the accelerating parameter.

It can be seen that the term $(\boldsymbol{1-\beta})$ tends to increase the values of $\boldsymbol{v\partial b}$ and $\boldsymbol{v\partial\theta}$**.** Thus it acts as an accelerating agent or accelerating parameter for the velocity terms $\boldsymbol{\partial w}$ and $\boldsymbol{\partial b}$**.** The same holds true for the bias $\boldsymbol{\partial\theta}$ too. β is called the friction parameter since the value of $\boldsymbol{\beta}$ is typically less than 1 and hence tends to reduce the velocity. Often $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are called the hyper parameters of learning [14]. The equation $\boldsymbol{b_{k+1} = b_k - \alpha v\partial w}$ represents the fact that that the gradient descent $\boldsymbol{b_k}$is smoothened out. The vertical acceleration is minimized with the aim to increase the vertical acceleration and in the process, the oscillations in the vertical acceleration or gradient are damped or smoothened out [15,16].

$(\boldsymbol{1-\beta})$ is the acceleration provided thereby increasing the momentum. Thus, this approach is called the gradient descent with momentum. The decrease in the overshoot or the vertical acceleration is responsible to impart momentum to the gradient by increasing the horizontal acceleration [17,18]. The concept of the acceleration of gradients is depicted in Fig. 2. It can be observed that as the vertical oscillations in gradient acceleration decrease, the horizontal acceleration increases in the GDM approach [19,20]. This suggests that the time of convergence and number of iterations would decrease in case of GDM as compared to the conventional GD [21].

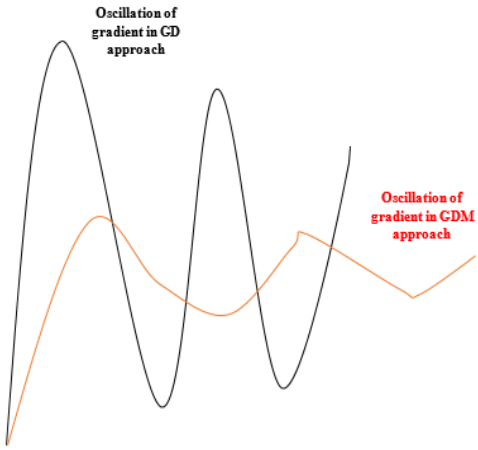

Fig. 2. The gradient of acceleration for GDM approach [22].

The pseudocode for the implementation of the GDM based approach is given below:

```
[NNcloudmodel,Tr] = traingdm(NNcloudmodel, trinp', trout');
forecast = sim(NNcloudmodel, testinp')';
%%
err=testout-forecast; % calculate error
mse=(mean(err.^2));
errpct = (abs(err)./testout)*100;
```

The execution time for a particular task is computed as:

$$T_{execution} = \sum_{i=1}^{n} A_k^i * \frac{DT_k^i}{VM_k} \quad (24)$$

Here, $T_{execution}$ denotes the task of execution, A_k^i= 1 if task 'i' is assigned to VM 'k'. Else A_k^i=0, DT_k^i is the amount of data task 'i' assigns to VM 'k', VM_k is the memory associated with VM 'k'

If $T_{execution}$ is known, the overall execution time or response time can be computed as:

$$T_{Response} = \sum_{i=1}^{m} T_{execution} \quad (25)$$

Here,

$T_{Response}$ is the response time, m is the total number of tasks.

In case, a VM is overloaded with execution of a particular task 'k', migration of task to another VM is employed based on the following equation [16,17]:

$$WC_r = WC_p - WC_T \quad (26)$$

Here, WC_r is the workload capacity of a particular VM, WC_p is the current or present workload of the VM and WC_T is the workload of the queued tasks for the VM

Migration of tasks form a particular VM is to be performed if the following relation is satisfied:

$$0 \leq WC_r \leq 1 \quad (27)$$

4. Results and Discussion

The cloud task scheduling has been implemented on the CloudSim 3.0 package in NetBeans. The parameters for the simulation of task scheduling are listed in Tables 1-3. The tables contain the data center details, task details and configuration details respectively. The evaluation parameters are the response time, the makespan, the number of iterations to reach convergence and mean square error.

Table 1. Data Center configuration details.

S. No.	Characteristics	Value
1	Allocation Policy	BAT+PSO
2	Architecture	X86
3	OS	windows
4	Hypervisor	Xen
5	VM Migration	Enabled
6	Time Zoned	10.0
7	Cost per BW	0.41
8	Number of data center	01
9	Number of processing unit	04
10	Storage capacity	1 TB
11	Total RAM	8GB

Table 2. Task details.

S. No.	Characteristics	Value
1	User	1
2	Task per minute	10
3	Avg length of the task	50,000 bytes
4	Avg Task file size	300 bytes
5	Avg Task file output size	300 bytes

Table 3. Configuration details.

S.No.	Characteristics	Value
1	Number of VM	10
2	Avg Image size	1000 bytes
3	RAM	512 MB
4	Bandwidth	1000 mbps
5	Scheduling Policy	Dynamic workload

It can be observed from the task details of tasks in Table 4 that the PSO has the maximum response time and the PSO+GDM based approach attains the least response time. The PSO+BAT has an intermediate response time. The response time is evaluated for three different tasks.

The task scheduling done is for the same application that has been scheduled on pre-created Virtual Machine, as is customary with SaaS offerings. These are the basic assumptions made for the task scheduling approach.

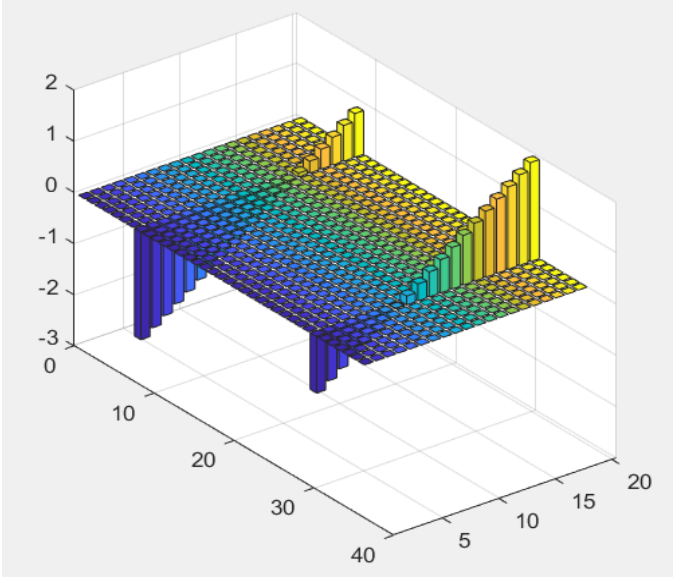

Fig. 3. Variation in the particle velocities in PSO.

Fig. 3 depicts the variation in the particle velocities for the PSO approach. The crests and the troughs show the variation prior to settling to the stable velocities.

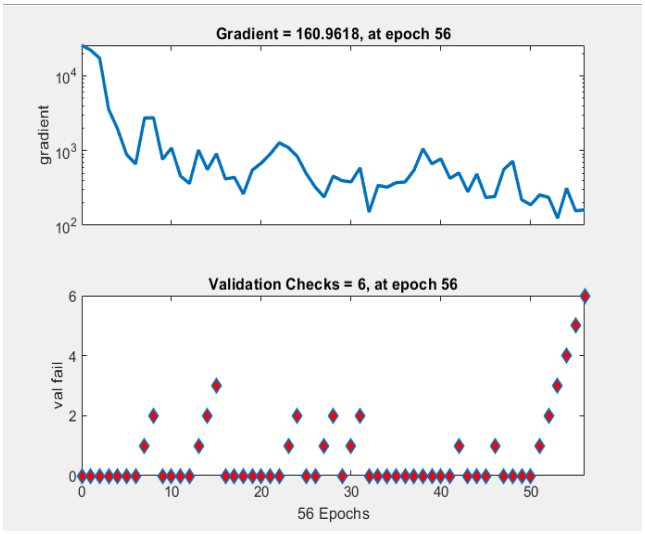

Fig. 4. Variation in the gradient and validation checks as a function of iterations.

The variation in the training states i.e. the gradient and the validation checks as a function of iterations is depicted in Fig. 4. The gradient can be observed to keep decreasing as the number of iterations keeps increasing. However, there are fluctuations in

the descent of the gradient.

The termination condition for the gradient descent with momentum algorithm is based on the following conditions:

1) The error becomes stable for the training process for 5-6 iterations, which leads the algorithm to decide to stop the training. The "val fail" or the validation fail condition stands for the condition that while the training is in progress, how many times the training error remains constant i.e. it is the number of iterations for which the error remains constant.
2) If the error doesn't reach stability, however the maximum number of iterations (considered 1000) here are over.

The fulfilment of either of the two conditions stated above governs the condition for termination of training.

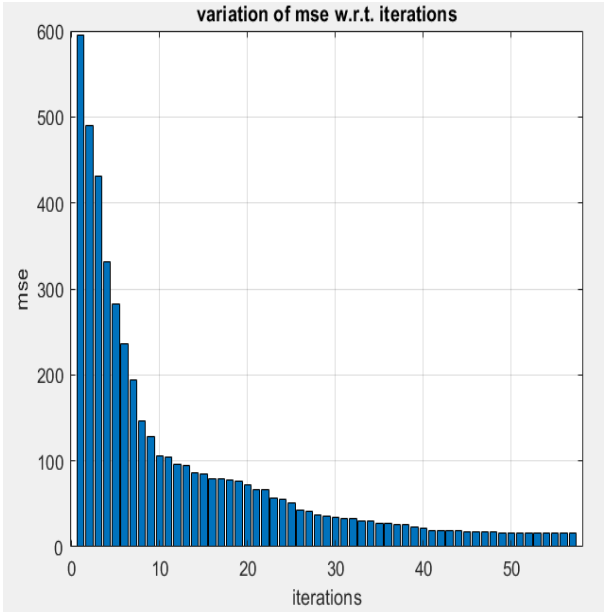

Fig. 5. Variation in the mse checks as a function of iterations.

Fig. 5 depicts the variation in the mse of the PSO+GDM approach as a function of iterations. It can be seen that as the iterations increase, the *mse* keeps decreasing monotonically prior to settling down around 50 iterations. 6 iterations are used as validation checks.

In general, the iterations for optimizing the cost function or objective function is terminated based on the following condition:

{

If iterations=Max iterations

Terminate iterations

Else if

Error stabilizes for validation checks
Terminate iterations
Else
Continue optimization
}

Table 4. Response time for different approaches.

Approach	VMs	RT Task1	RT Task2	RT Task3
PSO	10	5	5	5
BAT+PSO	10	2.8	1.9	1.9
GDM+PSO	10	1.9	1.7	1.8

Here, RT indicates the response time in ms.

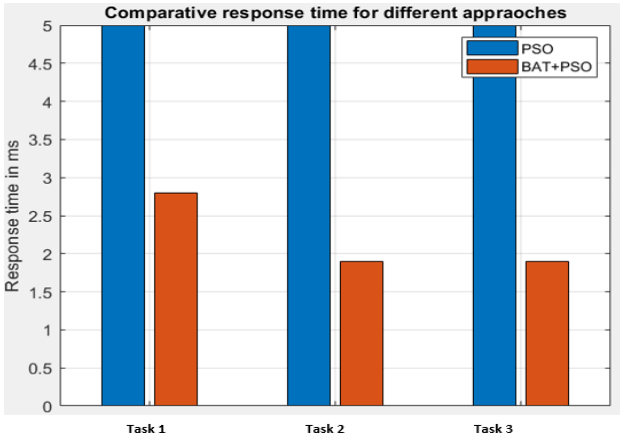

Fig. 6. Comparison of Response time for PSO and BAT+PSO approaches.

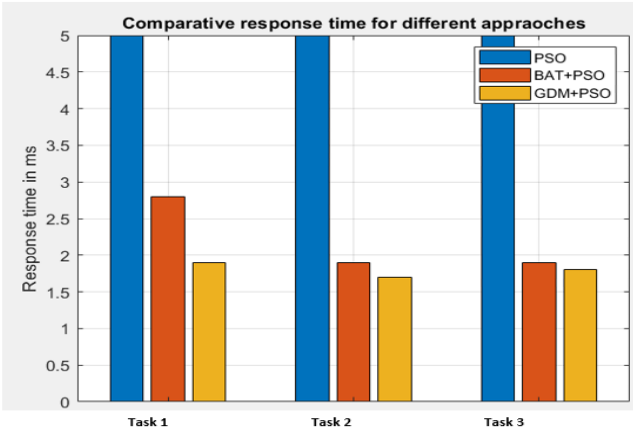

Fig. 7. Comparison of Response time for PSO, BAT+PSO and GDM+PSO approaches.

Fig. 6 depicts the response time for task scheduling using the PSO and BAT+PSO approaches. Fig. 7 depicts the response time for task scheduling using the PSO, BAT+PSO and GDM+PSO approaches. It can be clearly observed that the proposed approach comprising of the BAT and PSO algorithms takes lesser response time compared to the PSO alone for the tasks thereby indicating faster response, lesser latency and improved efficiency. Moreover, the least response time corresponds to the GDM+PSO approach.

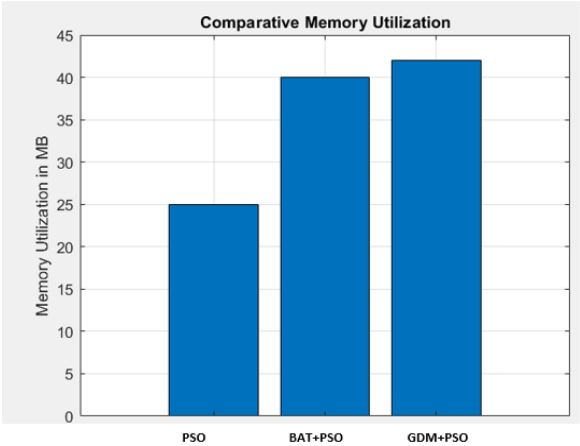

Fig. 8. Comparison Memory Utilization for PSO, BAT+PSO and PSO+GDM approaches.

Another important metric that governs the feasibility of a proposed approach is the memory utilization. The comparative memory utilization of the three different approaches is depicted in Fig. 8. It can be observed that the GDM+PSO approach has the maximum memory utilization which indicates that the GDM+PSO approach utilizes the memory resources most effectively.

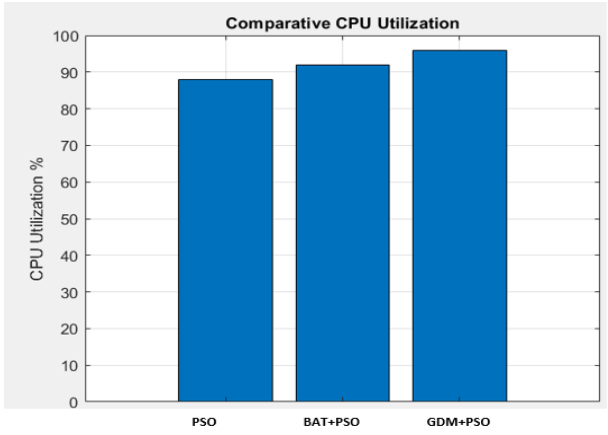

Fig. 9. Comparison CPU Utilization for PSO, BAT+PSO and PSO+GDM approaches.

Fig. 9 depicts the CPU utilization of the different approaches. It can be seen that the proposed PSO+GDM approach attains the maximum CPU utilization. It can be inferred from Figs. 8 and 9 that the CPU and memory utilization for the proposed GDM+PSO algorithm is the highest among the three heuristic approaches tested. While this indicates more computation capabilities of the machine dedicated to the algorithm, it is a trade-off to be considered to achieve higher speed or lesser latency for the system.

6. Conclusion

This paper proposes an evolutionary algorithm based approach for task scheduling in cloud computing. The benefit of using evolutionary algorithms is their ability of self-organization and adaptive nature without the strict necessity to be coded explicitly. The approaches explored in this paper are the PSO, the BAT-PSO and the GDM+PSO. The evaluation parameters have been chosen as the response time, the memory utilization, the CPU utilization and mean square error. It can be observed from the results that both BAT+PSO and GDM+PSO outperform the PSO applied individually in terms of the performance metrics. Moreover, the GDM+PSO performs better than the BAT+PSO. The present work focusses on task scheduling in cloud environments. The future enhancement of the proposed approach can be predictive models for cloud workload estimation. This would help cloud providers to manage constrained resources more effectively for cloud environments.

Acknowledgments

The authors acknowledge the creative suggestions of the faculty members of the department of Computer Science and Engineering, Medicaps University, Indore. The authors are also thankful to the management of Medicaps University for the conducive environment provided for research in the institute premises.

References

1. F. Juarez, J. Ejarque, and R. M. Badia, Future Generat. Computer Syst. **78**, 257 (2018). https://doi.org/10.1016/j.future.2016.06.029
2. L. Wang and E. Gelenbe, IEEE Transact. Cloud Computing **6**, 33 (2018). https://doi.org/10.1109/TCC.2015.2474406
3. M. Duggan, K. Mason, J. Duggan, A. Howley, and E. Barrett, Predicting Host CPU Utilization in Cloud Computing using Recurrent Neural Networks – *Proc. of 12th Int. Conf. for Internet Technology and Secured Transactions (ICITST), IEEE* (2018). https://doi.org/10.23919/ICITST.2017.8356348
4. N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning, *Proc. of 37th Int. Conf. on Distributed Computing Systems (ICDCS)* (2017). https://doi.org/10.1109/ICDCS.2017.123
5. L. Zuo, S. Dong, L. Shu, C. Zhu, and G. Han, IEEE Syst. J. **12** , 1518 (2016). https://doi.org/10.1109/JSYST.2016.2542251

6. Y. Hu, B. Deng, F. Peng, and D. Wang, Workload Prediction for Cloud Computing Elasticity Mechanism – *Proc. of Int. Conf. on Cloud Computing and Big Data Analysis (ICCCBDA)* (2016).
7. J. Xue, F. Yan, R. Birke, L. Chen, T. Scherer, and E. Smirni, PRACTISE: Robust Prediction of Data Center Time Series - *Proc. of 2015 11th Int. Conf. on Network and Service Management (CNSM)* (2015). https://doi.org/10.1109/CNSM.2015.7367348
8. F. Ramezani, J. Lu, and F. Hussain, Int. J. Parallel Programm. **42**, 739 (2014). https://doi.org/10.1007/s10766-013-0275-4
9. S. Sagnika, S. Bilgaiyan, and B. Mishra, Workflow Scheduling in Cloud Computing Environment using Bat Algorithm – *Proc. of 1st Int. Conf. on Smart System, Innovations and Computing* (2018) pp. 149-163. https://doi.org/10.1007/978-981-10-5828-8_15
10. L. Bottou, Stochastic Gradient Descent Tricks, Neural Networks: Tricks of the Trade - *Lecture Notes in Computer Science* (2012) **7700**, pp. 421-436. https://doi.org/10.1007/978-3-642-35289-8_25
11. J. Fliege, A. Vaz, and L. N. Vicente, J. Optimization Methods Software **34**, 949 (2019). https://doi.org/10.1080/10556788.2018.1510928
12. W. Liu, L. Chen, Y. Chen, and W. Zhang, IEEE Transact. Parallel Distributed Syst. **31**, 1754 (2020). https://doi.org/10.1109/TPDS.2020.2975189
13. J. Kumar and A. Singh, Future Generation Comput. Syst. **81**, 41 (2018). https://doi.org/10.4324/9780203732380
14. M. B. Gawali and S. K. Shinde, J. Cloud Computing **7**, ID 4 (2018). https://doi.org/10.1186/s13677-018-0105-8
15. S. Raghavan, P. Sarwesh, C. Marimuthu, and K. Chandrasekaran, Bat Algorithm for Scheduling Workflow Applications in Cloud – *Proc. on Int. Conf. on Electronic Design, Computer Networks & Automated Verification (EDCAV)* (2015). https://doi.org/10.1109/EDCAV.2015.7060555
16. M. Demirci, A Survey of Machine Learning Applications for Energy-Efficient Resource Management in Cloud Computing Environments – *Proc. of 14th Int. Conf. on Machine Learning and Applications (ICMLA)* (2015). https://doi.org/10.1109/ICMLA.2015.205
17. S. Abdelwahab, B. Hamdaoui, M. Guizani, and A. Rayes, IEEE Internet Things J. **1**, 276 (2014). https://doi.org/10.1109/JIOT.2014.2325071
18. S. Islam, J. Keunga, K. Lee, and A. Liu, Autonomic Scaling of Cloud Computing Resources using BN-based Prediction Models – *Proc. of 2nd Int. Conf. on Cloud Networking (CloudNet)* (2012).
19. E. Gelenbe, R. Lent, and M. Douratsos, Choosing a Local or Remote Cloud – *Proc. of 2nd Sympos. on Network Cloud Computing and Applications* (2012). https://doi.org/10.1109/NCCA.2012.16
20. M. Taheri, Z. Kamran, 2-Phase Optimization Method for Energy Aware Scheduling of Virtual Machines in Cloud Data Centers – *Proc. of Int. Conf. for Internet Technology and Secured Transactions* (2011).
21. S. Garg, K. Srinivasa, and R. Buyya, SLA-Based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter – *Proc. of Int. Conf. on Algorithms and Architectures for Parallel Processing* (2011) pp. 371-384. https://doi.org/10.1007/978-3-642-24650-0_32
22. https://www.coursera.org/lecture/deep-neural-network/gradient-descent-with-momentum-y0m1f (September 9, 2020)