Arabian Journal for Science and Engineering (2018) 43:7945–7960
https://doi.org/10.1007/s13369-018-3261-8

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Efficient Workflow Scheduling Algorithm for Cloud Computing System: A Dynamic Priority-Based Approach

Indrajeet Gupta[1] · **Madhu Sudan Kumar**[1] · **Prasanta K. Jana**[1]

Received: 26 April 2017 / Accepted: 12 April 2018 / Published online: 19 April 2018

Abstract
Workflow scheduling is one of the burning topics that has drawn enormous attention recently in the research community of cloud computing due to its wide applications in astronomy, physics, bioinformatics, health care and so on. This is a well-known NP-complete problem. It presents an interesting aspect of achieving minimum processing time of all the tasks and maximum resource utilization in cloud resources. Therefore, many algorithms have been developed for workflow scheduling. However, most of them consider a static priority of the tasks which is non-realistic for heterogeneous cloud computing environment. In this paper, we propose a workflow scheduling algorithm which considers dynamic priority of the tasks. The algorithm undergoes a process of min–max normalization followed by the calculation of the dynamic threshold to dispatch the tasks into one of the virtual machines. The algorithm is extensively simulated using various benchmark, scientific and real-life workflows. All the simulated results are compared with other four existing workflow scheduling algorithms. The simulated results confirm that the proposed algorithm lags behind all the four existing algorithms in terms of makespan and average cloud resource utilization. The simulation results are also validated through analysis of variance statistical test.

Keywords Cloud computing · Workflow scheduling · Min–max normalization · Makespan · Resource utilization

1 Introduction

Many scientific areas including bioinformatics, physics, astronomy and health care require executing complex jobs which are modeled as workflows. Such workflows are represented by a directed acyclic graph (DAG) in which a node denotes a task, and the data dependency is indicated by a directed edge between the two corresponding task nodes. However, the scientific workflows in the aforementioned areas are very large in size and complicated in structure. They require high computational power and storage for their processing. Therefore, various projects such as Pegasus [1,2], ASKALON [3] and GrADS [4] are designed for processing and management of the workflows in the grid environment. In the recent years, cloud computing [5–7] has gained enormous attention as the most suitable platform for executing large-scale scientific workflows due to the following reasons. It can provide unlimited resources by creating an illusion through virtualization technology. The users are charged on a pay-as-you-go basis because virtualized resources are dynamically provisioned as per the need of workflows and their deadline and budget. Moreover, cloud is more reliable than its counterparts such as grid and cluster computing. However, we require efficient schemes for scheduling large scientific workflows for their processing. In this paper, we propose an efficient scheme for workflow scheduling in a cloud environment.

The problem of workflow scheduling in cloud computing is to map each task to a suitable VM and to schedule the tasks for their execution. It is a challenging and a well-marked problem as NP-complete in nature. Therefore, various scheduling algorithms [5,6] have been developed for finding solution by fulfilling some optimal criteria. For examples, the works presented in [8–14] are designed to obtain considerable least makespan and maximum cloud resource utilization. The algorithms in [12,13] are designed to achieve

✉ Indrajeet Gupta
indrajeet7830@gmail.com

Madhu Sudan Kumar
mdhsdnkumar88@gmail.com

Prasanta K. Jana
prasantajana@yahoo.com

[1] Department of Computer Science and Engineering, Indian Institute of Technology (ISM), Dhanbad, Dhanbad 826004, India

cost optimality. The works reported in [15] emphasize on the energy efficiency. However, most of these algorithms are developed for independent tasks and thus they are not applicable for the workflow applications.

In this paper, we propose an algorithm which is applicable for workflow scheduling. The algorithm is based on min–max normalization on estimated computation time (ECT) of all the tasks. However, it uses dynamic threshold value for dividing all the tasks into small batch and large batch in order to obtain task to VM mapping with the application of minimum completion time algorithm [16]. The proposed algorithm is extensively simulated using various scientific workflows [2], and the simulation results are compared with four existing algorithms, namely heterogeneous earliest finish time (HEFT) [9], dynamic level scheduling (DLS) [10], min–min [11] and max–min [11]. The overall workflow execution time (i.e., makespan) and average cloud utilization both are considered as the performance parameters for the proposed algorithm.

Recently, Panda and Jana [13,17] have presented various task scheduling algorithms. The basic idea of our proposed algorithm is similar to them. However, it has the following notable differences and advantages. (1) We consider the dynamic threshold value to divide the tasks into two different batches in contrast to a static threshold as used in [13,17]. (2) To make more realistic our proposed algorithm incorporates data transfer time between the tasks which is not considered by their approach. (3) They have shown the simulation results for independent tasks which is not applicable for workflow scheduling.

The contributions of this paper can be summarized as follows:

- Development of a workflow scheduling algorithm based on min–max normalization
- Design of a dynamic threshold for categorizing the task in the ready queue
- Rigorous simulation on different benchmark, scientific and real-life workflows
- Performance evaluation by considering makespan and average cloud utilization as the performance parameters
- Comparison of the simulation results with four well-known existing algorithms to show the effectiveness of the proposed algorithm
- Validate the results by performing a hypothetical test, called ANOVA [18].

The rest of the paper is outlined as follows: Sect. 2 describes few notable research work related to workflow scheduling problems. Section 3 states workflow scheduling problem formulation with the cloud model functionality. Section 4 explains the proposed work with an illustrative example followed by Sect. 5, in which experimental results and performance evaluation with comparisons of proposed algorithm with the existing works are well loaded. Finally, Sect. 6 summarizes the paper.

2 Related Work

Numerous research works [19–25,27,28] have been surveyed and proposed for workflow scheduling in cloud environment followed by both centralized and multi-agent distributed approaches. The workflows can be categorized in two main categories, (1) data-intensive and (2) compute-intensive workflows. The communication-to-computation ratio (CCR) value is applied to categorize such workflows. When the value of CCR is very low, a workflow is fallen under compute-intensive otherwise, it is data-intensive [21,24,26,27]. The algorithm which we propose in this paper is capable to handle both these types, i.e., compute-intensive and data-intensive workflow applications. Due to NP-complete nature, various heuristic and meta-heuristic-based algorithms have been carried out for workflow scheduling. Most of the algorithms have certain objectives such as minimizing cost, makespan, energy consumption and maximizing reliability and resource utilization [5,8,28–31]. Yu et al. [32] presented a task scheduling algorithm which works in two phase, and a three-level cloud environment is taken as the computing environment. This scheduling algorithm is a hybridization of the traditional opportunistic load balancing (OLB) and min–min load-balanced scheduling strategy. However, this method is limited for the three-level cloud architecture. Another noticeable work has been proposed by Ming and Li [25]. They tried to recover the waiting time between the short tasks by applying max–min scheduling scheme.

Cloud computing is itself a multi-agent-based system, which creates an environment for providing various services. The workflow management system can also be benefited by the agent-based paradigm over the centralized system to solve the complex problems. A multi-agent system (MAS) is a group of loosely coupled network of software agents of different capabilities which interact to each other to solve the given problem. The detailed review and applications about the multi-agent systems are well described in [33]. In [8], Bochenina et al. proposed three workflow scheduling algorithms for multiple workflows. In their work, a soft deadline-based scheduling scheme have been introduced for mapping the tasks of multiple workflows within the various different deadlines by considering static set of VMs with prior known empty time windows. Another well-known cloud platform OpenNebula [30] follows the task rank scheduling policy, and the match making scheduler is used in this scheduling scheme. Here, highest rank tasks are to be mapped on the suitable VM.

Nimbus [34] searches virtual resources having the minimum percentage of unallocated random access memory. After that it keeps the resource in sleep mode if the resource has not been used for a long time, so that the resource may utilize the proper power consumption. Therefore, in case of energy-efficient scheduling it is one remarkable work. However, it provides poor makespan. Freund et al. [35] drawn the advantage of min–min and max–min as the computational intensive algorithms, both were applied by Ibarra and Kim [26]. Afterwards, Braun [36] displayed that min–min gives quite satisfactory makespan, whereas the resource utilization is produced by max–min algorithm as compared to other scheduling algorithms. Recently, Li et al. [12] have proposed cloud min–min scheduling which is the extension of the popular min–min algorithm. This is first work which belongs to heterogeneous multi-cloud environment for bag of tasks. In the similar fashion, Panda and Jana [17] have proposed CMAXMS (cloud max–min scheduling) and CMINMAXS (cloud min–min max–min scheduling), respectively. However, these algorithms have considered task to cloud mapping rather than mapping of the tasks to the virtual machines. Hence, their method may not be applicable for the workflow scheduling in cloud computing.

Some algorithms [8,9,12,13] are presented for cloud environment, but they consider static workflows, and therefore, they are not suitable for dynamic workflow scheduling. In the case of dynamic scheduling, the information of workflows and cloud resources are not given in advance and it is to be available during the scheduling of the workflows. Since it is one of the core concern in the cloud computing that its resources are elastic, a dynamic scheduling makes more sense in the realistic scenario and the remarkable work has been done by Salah et al. [37]. Their work has been designed to achieve the best suitable cloud resources while the processing of user request as per the every service level objectives (SLOs) satisfaction. This work uses the Markov chain model efficiently for prediction of the cloud resources to fulfill the user-defined SLOs. There are many other algorithms that deal with the dynamic scheduling of workflows. However, this is not concerned with the proposed technique as it deals with static scheduling where all the tasks and availability of VMs are priori.

This is noteworthy that the well-known algorithms, i.e., HEFT [9], Min–Min, Max–Min and DLS are not fully able to fill the idle time slot of computing resources, i.e., VMs. The other task replication-based task scheduling algorithms ever demand the infinite resource infrastructure. Therefore, the improvement for utilization of VMs under heterogeneous cloud infrastructure is always desirable. So we present a new dynamic priority list-based workflow scheduling algorithm for heterogeneous cloud.

3 Cloud Model and Problem Formulation

This section describes the cloud model which is used throughout in this paper. The problem formulation and some preliminaries are well organized step by step.

3.1 Cloud Model

We consider a cloud model which is applicable with the predefined assumptions for workflow scheduling. In this model, the cloud servers offer various VM instances of dissimilar capabilities. The VM manager takes the all responsibilities of VM creation and deletion as per the global cloud manager requirement. The workflow manager is the module to split the workflow applications into the task pool and also send the task dependencies information in the form of adjacency list to the global cloud manager. The global cloud manager works as the cloud scheduler. It keeps track of the status of VMs which are deployed in all the host systems on cloud servers. When a user application (workflow) is submitted to a workflow manager, it splits up that workflow into the set of tasks or cloudlets and submits to global cloud manager deployed separately in the cloud server physically. Then, global cloud scheduler fetches the tasks from the ready task list and then schedules the task on the scalable VM pool as per the scheduling strategy. If the tasks are assigned to the VMs those are deployed on the same cloud server, then the tasks do not require any data transfer time; otherwise, there will be a data transfer time required to handover the task from one cloud server to another cloud server.

3.2 Workflow Scheduling Problem Formulation

3.2.1 Workflow Application

A workflow $D_g = (T, E)$ is symbolized by a directed acyclic graph (DAG) where $T = \{T_1, T_2, \ldots, T_n\}$ is the set of n task nodes and E is the set of edges. An edge from T_p to T_s indicates that the task T_s cannot be started until T_p is completed and this is denoted by $T_p \prec T_s$. The task T_p is the predecessor, and T_s is the successor.

Definition 3.1 In a workflow application, a task node without any ancestor task is known as entry task and a node without any successor is known as exit task node. If a workflow has multiple entry nodes, a pseudo entry task is added to connect it with all the entry nodes by means of virtual links having zero weight (i.e., transfer time). The computation time of this pseudo entry task is also assumed as zero. Similarly, in case of multiple exit nodes, we introduce a pseudo exit task to connect it with all the exit nodes by applying the same assumption as that of the pseudo entry task. The data

transfer time between task nodes can be represented by an upper triangular matrix denoted by DT as follows:

$$DT = \begin{array}{c} \\ T_1 \\ T_2 \\ \vdots \\ T_n \end{array} \begin{array}{c} \begin{array}{cccc} T_1 & T_2 & \dots & T_n \end{array} \\ \begin{bmatrix} 0 & DT_{1,2} & \dots & DT_{1,n} \\ 0 & 0 & \dots & DT_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \end{array} \quad (1)$$

where the element DT_{ij}, $1 \leq i \leq n$, $1 \leq j \leq n$ represents the time for data transfer between the task T_i and the task T_j.

3.2.2 Cloud Resources or VM Instances

A cloud service provider (CSP) offers a set of VM instances of different capability which are suitable for execution of a given workflow. Let $CS = \{CS_1, CS_2, CS_3, \dots, CS_M\}$ be the available set of cloud servers which deploy m number of VMs. These VMs are responsible for execution of the tasks of the given workflow. Note that, the number of active VMs is changeable from one cloud server to another cloud server as per the deployment capacity of cloud server. It is also assumed that the data transfer time among two VMs is negligible if both the VMs are deployed on the same cloud server.

3.2.3 Estimated Computation Time (*ECT*)

A task may take different execution time on different VMs. This estimated computation time is represented by a matrix called ECT which is defined as follows:

$$ECT = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_n \end{array} \begin{array}{c} \begin{array}{cccccccc} \overbrace{VM_1 & \dots & VM_{|CS_1|}}^{CS_1} & & \dots & \overbrace{VM_{\alpha+1} & \dots & VM_{\alpha+|CS_M|}}^{CS_M} \end{array} \\ \begin{bmatrix} ET_{1,1} & \dots & ET_{1,|CS_1|} & \dots & ET_{1,\alpha+1} & \dots & ET_{1,\alpha+|CS_M|} \\ ET_{2,1} & \dots & ET_{2,|CS_1|} & \dots & ET_{2,\alpha+1} & \dots & ET_{2,\alpha+|CS_M|} \\ ET_{3,1} & \dots & ET_{3,|CS_1|} & \dots & ET_{3,\alpha+1} & \dots & ET_{3,\alpha+|CS_M|} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ ET_{n,1} & \dots & ET_{n,|CS_1|} & \dots & ET_{n,\alpha+1} & \dots & ET_{n,\alpha+|CS_M|} \end{bmatrix} \end{array} \quad (2)$$

where α is $\sum_{k=1}^{M-1} |CS_k|$ and cell $ET_{i,j}$, $1 \leq i \leq n, 1 \leq j \leq |CS_M|$ represents the estimated computation time of the task T_i on VM_j. This time can be estimated by using different routine techniques (e.g., empirical and analytical modeling [31] and historical data [4,38]).

Before moving ahead in the problem formulation, the following assumptions are considered throughout the paper.

1. The runtime of the tasks in the given workflows is known priorly.
2. Single task from the task pool must be executed on a single VM at a time. Neither the partial execution nor the duplication of the tasks is allowed.
3. There should not be any preemption of the task from a VM once the task is allotted to VM.

3.2.4 Cloud Makespan

The makespan (MS) is the overall processing time of a workflow application. So, this is the elapsed time from the starting of executing entry task till the end of the exit task. Note that, all the tasks need to be executed on available VMs of the cloud servers. Let $MS(CS_k)$ be the makespan corresponding to all the tasks executed on the VMs of the cloud server CS_k, $1 \leq k \leq M$ and $MS(VM_j)$ be the makespan of the tasks executed on the virtual machine say, VM_j, $1 \leq j \leq |CS_k|$ (assuming that VM_j is deployed in cloud CS_k). Then the individual makespan on the cloud server CS_k, $1 \leq k \leq M$ can be mathematically expressed as follows:

$$MS(CS_k) = max(MS(VM_j)), 1 \leq j \leq |CS_k|. \quad (3)$$

Therefore, the overall makespan on all the cloud servers for a given workflow application is given by

$$MS = max(MS(CS_k)), 1 \leq k \leq M. \quad (4)$$

3.2.5 Average Cloud Utilization (CU)

Time consumption rate of any VM with respect to the cloud server on which the VM is deployed is known as the VM utilization. The proportion between the VM makespan and the entire engaged (busy) time (i.e., makespan) of the cloud server CS_k on which the VM is deployed [17]. Mathematically, it can be expressed as follows:

$$U(VM_j) = \frac{MS(VM_j)}{MS(CS_k)}, \quad 1 \leq j \leq |CS_k| \quad (5)$$

where $U(VM_j)$ denotes the utilization of VM_j. Therefore, the utilization of the cloud server can be defined as the average utilization of all the deployed VMs on same cloud server. It is represented mathematically as:

$$U(CS_k) = \frac{1}{|CS_k|} \sum_{j=1}^{|CS_k|} U(VM_j) \quad (6)$$

where $1 \leq j \leq |CS_k|$, $1 \leq k \leq M$.

In the above-mentioned equation, $U(CS_k)$ denotes the utilization of cloud server CS_k. Similarly, the average utilization of all cloud servers defined as average cloud utilization.

$$CU = \frac{1}{M} \sum_{k=1}^{M} U(CS_k), \quad 1 \leq k \leq M. \quad (7)$$

Problem Statement The problem is to generate a schedule plan which maps all the tasks of the given workflow on m

active VMs deployed on M cloud servers such that makespan is minimized and average cloud utilization is maximized.

3.3 Notations and Preliminaries

Let us now define the other notations/terminologies that are useful to describe our proposed algorithm. All such notations along with their descriptions are depicted in Table 1.

1. Estimated computation time ($ET_{i,j}$): It is the estimated computation time of task T_i on VM_j as defined earlier in Eq. 2.
2. Earliest start time ($EST_{i,j}$): The $EST_{i,j}$ is the time of a task T_i, when a task can begin its execution on a VM_j. It searches for the most extreme (max) of the followings: (1) minimum accessible time of VM_j (i.e., $Avail_j$) and (2) maximum of the addition of the completion time of all the antecedent tasks of a given task T_i, where $1 \leq i \leq n$ and the information (data) transfer time from the parent tasks to the current task T_i. Mathematically, $EST_{i,j}$ is formulated as follows:

$$EST_{i,j} = max\left\{Avail_j, \max_{p_i \in predr(i)}\left\{ACT_{p_i} + DT_{p_i,i}\right\}\right\} \tag{8}$$

where a function max seeks the maximum of all the values, p_i denotes a predecessor task, $predr(i)$ consists a set the parent (predecessor) tasks of current task T_i, ACT_{p_i} represents the time of actual completion for a predecessor task T_{p_i}, $p_i \in predr(i)$ and $DT_{p_i,i}$ denotes the data transfer time from a predecessor task T_{p_i} to current task T_i.

Remark 3.1 For all the entry tasks the $EST_{i,j}$ will be zero, i.e., $EST_{\text{entry},j} = 0$.

Remark 3.2 The data transfer time from a predecessor task (T_{p_i}) to the current task T_i will not be considered if both the tasks are mapped to the same VM or same cloud server.

3. Earliest finish time ($EFT_{i,j}$): The $EFT_{i,j}$ of a task T_i on VM_j is calculated by summing of the earliest (immediate) start time and estimated computation time of the task T_i on the VM_j which is formulated as follows:

$$EFT_{i,j} = EST_{i,j} + ET_{i,j} \tag{9}$$

4. Actual completion time ($ACT_{i,j}$): It is the absolute time by which VM_j completes the task T_i.
5. Normalized estimated completion time ($N\_ECT_{i,j}$): $N\_ECT(i, j)$ is formulated as follows:

$$N\_ECT_{i,j} = \left(\frac{ECT_{i,j} - min\{ECT_{i,j}\}}{max\{ECT_{i,j}\} - min\{ECT_{i,j}\}}\right) \tag{10}$$

where $1 \leq i < n,\ 1 \leq j \leq m$

Initially, at the level 1 of the workflow, the normalized $ECT_{i,j}$ is calculated by Eq. 10. After that $N\_ECT_{i,j}$ updates as

Table 1 List of notations

Notation	Description
l_i	i^{th} level of workflow D_g
QR_{l_i}	Ready task list at level l_i of workflow D_g
$\|QR_{l_i}\|$	Total no. of tasks at level l_i of workflow D_g
$ET_{i,j}$	Estimated computation time of task T_i on VM_j
$EST_{i,j}$	Earliest start time of task T_i on VM_j
$EFT_{i,j}$	Earliest finish time of task T_i on VM_j
$DT_{i,j}$	Data transfer time from task T_i to task T_j
n	Numbers of task node in a workflow D_g
m	Numbers of VMs used for scheduling
M	Total number of cloud servers (CS)
$ACT(i)$	Actual completion time of task T_i
$COMP\_EFT()$	Function for compute estimated finish time
CCR	Average communication-to-computation ratio of workflow D_g
$Avail_j$	Available time of VM_j
$Temp\_ECT_{i,j}$	Temporary $ECT_{i,j}$ matrix after mapping a task T_i on VM_j in QR_{l_i} at level l_i of D_g
$NTemp\_ECT_{i,j}$	Normalized $Temp\_ECT_{i,j}$ matrix in ready queue QR_{l_i} at each level l_i of D_g
$Thrsh(d_{T_i})$	Dynamic Normalized threshold value threshold
B_large	Large batch of tasks at each level l_i of D_g
B_small	Small batch of tasks at each level l_i of D_g

$N\_Temp\_ECT_{i,j}$ because if any single task T_i is assigned to any available VM_j then the previous $ECT_{i,j}$ needs to be updated as $Temp\_ECT_{i,j}$.

Remark 3.3 Similarly, $N\_ECT_{i,j}$ also updates as corresponding $N\_Temp\_ECT_{i,j}$ at each level l_i of workflow D_g.

6. Dynamic normalized threshold value ($Thrsh(d_{T_i})$): $Thrsh(d_{T_i})$ of task T_i is the ratio between maximum computation time of task T_i on m VMs and maximum data transfer time $DT_{p_i,i}$ at task T_i from all of its predecessors' task set $\{T_p\}$. If the task node T_i do not have any predecessor task in predecessors tasks set $\{T_p\}$, then data transfer time will be zero. Mathematically, $Thrsh(d_{T_i})$ is calculated as follows:

$$Thrsh(d_{T_i}) = \begin{cases} \frac{\sum_{i \in \lfloor QR_{l_i} \rfloor} \underset{j \in m}{Max}(ECT_{i,j})}{\sum_{i \in \lfloor QR_{l_i} \rfloor} \underset{p_i \in \{T_p\}}{Max}(DT_{p_i,i})} & \text{if } T_p \neq \Phi \\ \text{Fixed} & \text{if } T_p = \Phi \end{cases} \tag{11}$$

where $1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq p_i \leq |T_p|$

Remark 3.4 The $Thrsh(d_{T_i})$ is totally independent from $Temp\_ECT_{i,j}$. It is calculated on the basis of initial $ECT_{i,j}$ at each level of workflow D_g.

7. Avg. communication-to-computation ratio (CCR): It is the ratio of average data transfer time (communication cost) to the average computation cost of any DAG. It is the property of the DAG or workflow. On the basis of CCR value, the DAG is recognized as either compute-intensive or data-intensive.

4 Proposed Algorithm

The basic concept of the proposed algorithm is as follows. We first divide all the tasks of the given workflow into the bag of tasks (BOTs) at each level. Now we start iterating the schedule of the tasks with the first BOT as follows. The ECT value for all the tasks in the BOT is normalized using min–max normalization. Then, we apply a dynamic threshold value to divide all the tasks of the BOT into two batches. Finally, we perform scheduling by applying minimum completion time algorithm on each of batches, the large batch first followed by the small batch.

The proposed algorithm works in two phases at each level l_i of workflow D_g without assigning the task's priority. In the first phase, it normalizes the $ECT_{i,j}$ of each task T_i at each level l_i. It then calculates the threshold value of each task as per Eq. 11. The maximum normalized value of $\{N\_ECT_{i,j}\}$ is compared with the threshold value $Thrsh(d_{T_i})$ to divide all the tasks at each level into small and large batches. This is done from the entry task node to the exit task node in top to bottom fashion. In the second phase, the tasks are assigned to the VMs by choosing minimum execution time from the $ECT_{i,j}$. After task assignment, further the $ECT_{i,j}$ needs to be updated as $Temp\_ECT(i, j)$. This process continues until all the tasks are scheduled. In second phase, EST and EFT are calculated of that task which has to be scheduled as per the algorithm (refer Eqs. 8, 9). The detailed description of the above-mentioned method is shown in pseudo code from Algorithm 1 through Algorithm 4.

Algorithm 1 : NMMWS Workflow Scheduling

Input- $D_g(T, E)$ workflow consisting of n task nodes

Output- Schedule plan generation S of workflow $D_g(T, E)$ on all available m VMs deployed on M cloud servers

```
Read the estimated computation time ECT and data transfer time
    DT of a given workflow application D_g(T, E) where T_i∈T and
    DT_{i,j}∈E
while there are unscheduled tasks in the task pool do
  for each level l_i of D_g(T, E) do
    Call GENERATE-READY-TASKS-LIST
        (n, ECT_{i,j})
    Call COMP_EFT(ECT_{i,j} of QR_{l_i}, DT_{i,j})
    CallNORM_PARTITION (ECT_{i,j}, N_ECT_{i,j}, Thrsh(d_{T_i}))
    if B_large is not empty then
      find min(EFT(i,j))
      taskVMmap(T_i)=VM_j
      ACT_{i,j} = EFT_{i,j}
      EST_{i,j} = 1
      remove T_i from B_large
    else
      find min(EFT_{i,j})
      taskVMmap(T_i)=VM_j
      ACT_{i,j} = EFT_{i,j}
      EST_{i,j}=1
    end if
  end for
end while
```

Algorithm 2 :
GENERATE-READY-TASKS-LIST$(n, ECT_{i,j})$

```
while there is unscheduled task in workflow D_g(T, E) do
  B_large of tasks is empty
  for each task T_i do
    if ECT_{i,j} then
      Add this T_i to QR_{l_i}
    end if
  end for
end while
```

Algorithm 3 : *COMP_EFT*($ECT_{i,j}$ of QR_{l_i}, $DT_{i,j}$)

```
1  for each task T_i in ready task list QR_{l_i} do
2    if T_i has no parent task T_p then
3      ECT_{i,j} = Avail_j + ET_{i,j}
4    else
5      for each VM_j and T_p in parent task of T_i do
6        if taskVMmap(T_p) is not equal VM_j then
7          if ECT_{i,j} <
   max{Avail_j, max{ACT_{p_i}+DT_{p_i,i}}} then
8            ECT_{i,j} =
   max{Avail_j, max{ACT_{p_i}+DT_{p_i,i}}}
9          end if
10       else
11         if ECT_{i,j} < max(Avail_j, ACT_{p_i}) then
12           ECT_{i,j} = max(Avail_j, ACT_{p_i})
13         end if
14       end if
15     end for
16   end if
17 end for
```

Algorithm 4 :*NORM_PARTITION* ($ECT_{i,j}$, $N\_ECT_{i,j}$, $Thrsh(d_{T_i})$)

```
1  for each task T_i in ready task list QR_{l_i} and VM_j do
2    Find min(ECT_{i,j})
3    Find max(ECT_{i,j})
4    Find the normalized ECT_{i,j} as N_ECT_{i,j}
                                   /* using Eq. 10 */
5    Find the Thrsh(d_{T_i})       /* using Eq. 11 */
6    if max {N_ECT_{i,j}} ≥ Thrsh(d_{T_i}) then
7      add task T_i to batch B_large
8    else
9      task T_i go to batch B_small
10   end if
11 end for
```

Lemma 1 *Time complexity of procedure GENERATE-READY-TASKS-LIST* $(n, ECT_{i,j})$ *is* $O(n^2)$.

Proof There are n task nodes which are to be scheduled on m VMs. Before scheduling all the task nodes at each level of the DAG l_i are maintained by the ready task list QR_{l_i}. So the while loop from Step 1 through Step 8 requires $O(n)$ time in worst case and inner for loop from Step 3 through Step 7 iterates by $O(n)$ times. Therefore, the time complexity of the procedure *GENERATE-READY-TASKS-LIST* is $O(n^2)$. □

Lemma 2 *Time complexity of COMP_EFT* $(ECT_{i,j}, DT_{i,j})$ *is* $O(nm)$.

Proof For the n separated tasks of the DAG, the for loop iterates Step 1 through Step 17 n times and the inner for loop iterates Step 5 through Step 15 m times. Here m is the total available VMs which are deployed on M cloud servers. So *COMP_EFT* $(ECT_{i,j}, DT_{i,j})$ requires $O(nm)$ time complexity. □

Lemma 3 *The procedure NORM_PARTITION has the Time complexity of* $O(n^2m)$.

Proof Let n be the total task nodes in a DAG application and m is the number of all active VMs which are deployed at M cloud servers. In this procedure, from Step 1 to Step 11 the for loop iterates (nm) times. Then Step 2 and Step 3 both require $O(nm)$ time and Step 4 and Step 5 both require $O(n)$ time complexity. Therefore, procedure *NORM_PARTITION* takes $O(n^2m)$ time complexity. □

Lemma 4 *The overall time complexity of the proposed algorithm NMMWS is* $O(n^3ml)$.

Proof Let n be the number of total task nodes with e connecting edges, l is the count of levels in the given workflow by considering m active VMs which are deployed at M cloud servers. In the algorithm, Step 2 and Step 3 require $O(n+e)$ and $O(l)$ time, respectively. Consequently, Step 4 demands $O(n^2)$ time for generating ready task list for n number of tasks in the worst case. Step 5 requires $O(nm)$ time for computing estimated completion time of each task at each level of l_i of the DAG. Step 6 requires $O(n+e)$ time. The for loop from Step 3 to Step 19 iterates n^2m times. It takes $O(n^2m)$ time accordingly. Here, the while loop from Step 2 to Step 20 iterates n times. Hence, the overall time complexity of the of the proposed workflow scheduling algorithm *NMMWS* is $O(n^3ml)$. □

4.1 An Illustration

We consider the Montage DAG as an example workflow as figured in Fig. 1. It orientates as 15 tasks, i.e., $T = \{T_1, T_2, \ldots, T_{15}\}$ with 7 levels and 24 edges among the tasks. The Montage workflow follows the hybrid structure with parallelism and pipelining. The data transfer time matrix $DT_{i,j}$ denotes the file transfer time among the tasks which represents the data transfer time between the tasks as shown in Table 2. The estimated computation time ECT matrix represents computation time of each task on the 4 virtual machines which are deployed on two cloud servers as shown in Table 3.

In Fig. 1, level-1 of the workflow has three task nodes in ready list QR_{l_1}. They are T_1, T_2 and T_3, and all these three tasks have the possibility to map on all the 4 available VMs at both the cloud servers. So at level-1, tasks T_1, T_2 and T_3 consist the $Temp\_ECT(i, j)$ as follows:

$$Temp\_ECT_{i,j} = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \end{array} \begin{pmatrix} VM_1 & VM_2 & VM_3 & VM_4 \\ 17 & 14 & 13 & 22 \\ 14 & 17 & 14 & 16 \\ 19 & 17 & 16 & 12 \end{pmatrix}$$

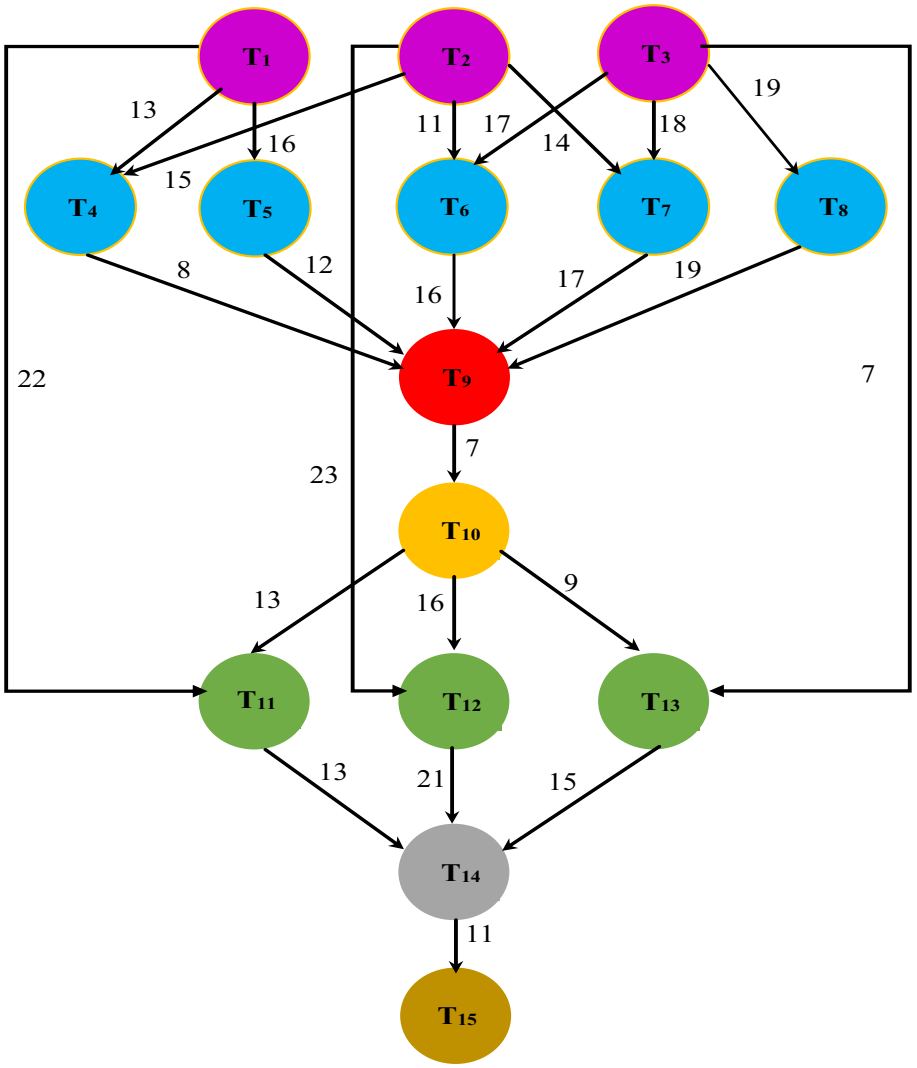

Fig. 1 A Montage workflow of 15 task nodes

Table 2 Data transfer time matrix with 24 dependency edges and 15 tasks of Montage workflow

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
T_1	0	0	0	13	16	0	0	0	0	0	22	0	0	0	0
T_2	0	0	0	15	0	11	14	0	0	0	0	23	0	0	0
T_3	0	0	0	0	0	17	18	19	0	0	0	0	17	0	0
T_4	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0
T_5	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0
T_6	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0
T_7	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0
T_8	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0
T_9	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0
T_{10}	0	0	0	0	0	0	0	0	0	0	13	16	9	0	0
T_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0
T_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	21	0
T_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0
T_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11
T_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3 An ECT matrix for Montage workflow DAG with 15 tasks and 4 virtual machines on 2 cloud servers

		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
CS_1	VM_1	17	14	19	13	19	13	15	19	13	19	13	15	18	20	11
	VM_2	14	17	17	20	20	18	15	20	17	15	22	21	17	18	18
CS_2	VM_3	13	14	16	13	21	13	13	13	13	16	14	22	16	13	21
	VM_4	22	16	12	14	15	18	14	18	19	13	12	14	14	16	17

According to Eq. 10, normalized $Temp\_ECT_{i,j}$ will be as follows:

$$N\_Temp\_ECT_{i,j} = \begin{matrix} \\ T_1 \\ T_2 \\ T_3 \end{matrix} \begin{pmatrix} VM_1 & VM_2 & VM_3 & VM_4 \\ 0.50 & 0.20 & 0.10 & 1.00 \\ 0.20 & 0.50 & 0.20 & 0.40 \\ 0.70 & 0.50 & 0.40 & 0.00 \end{pmatrix}$$

In DAG example, tasks T_1, T_2 and T_3 have no predecessors tasks so there will be no data transfer time at tasks T_1, T_2 and T_3 from its predecessors tasks. If $DT_{p_i,i}$ is zero, then the threshold $Thrsh(d_{T_i})$ will be also zero as per Eq. 11. Therefore, the $Thrsh(d_{T_i})$ is fixed as 0.99 for such type of situations. Now we compare $Thrsh(d_{T_i})$ with max element of $N\_Temp\_ECT_{i,j}$. $MaxN\_Temp\_ECT_{i,j}$ = 1.00 in the $N\_Temp\_ECT_{i,j}$ matrix at level-1 for task T_1 compared with $Thrsh(d_{T_i})$ which is fixed as 0.99. $Thrsh(d_{T_i}) < MaxN\_Temp\_ECT_{i,j}$; therefore, task T_1 is placed in large batch (b_large). By calculating EST and EFT as per Eqs. 8 and 9 task T_1 is mapped on the VM_3 at cloud server 2. Now updated $Temp\_ECT(i, j)$ for tasks T_2 and T_3 will be as follows:

$$Temp\_ECT_{i,j} = \begin{matrix} \\ T_2 \\ T_3 \end{matrix} \begin{pmatrix} VM_1 & VM_2 & VM_3 & VM_4 \\ 14 & 17 & 13+14 & 16 \\ 19 & 17 & 13+16 & 12 \end{pmatrix}$$

and corresponding Normalized $Temp\_ECT_{i,j}$ will be as follows:

$$N\_Temp\_ECT_{i,j} = \begin{matrix} \\ T_2 \\ T_3 \end{matrix} \begin{pmatrix} VM_1 & VM_2 & VM_3 & VM_4 \\ 0.1176 & 0.2941 & 0.8824 & 0.2353 \\ 0.4118 & 0.2941 & 1.00 & 0.00 \end{pmatrix}$$

$MaxN\_Temp\_ECT_{i,j}$ in the updated $N\_Temp\_ECT_{i,j}$ matrix, is also 1.00 for task T_3 also compared with $Thrsh(d_{T_i})$ which is fixed as 0.99. Here, also $Thrsh(d_{T_i}) < MaxN\_Temp\_ECT_{i,j}$; therefore, task T_3 is also placed in large batch (b_large). Similarly, after finding EST and EFT as per Eqs. 8 and 9 the task T_3 is mapped on the VM_4 at cloud server 2. Now at level 1 of DAG, task T_2 has the updated $Temp\_ECT_{i,j}$ as follows:

$$\begin{matrix} \\ T_2 \end{matrix} \begin{pmatrix} VM_1 & VM_2 & VM_3 & VM_4 \\ 14 & 17 & 13+14 & 16+12 \end{pmatrix}$$

Table 4 The tasks to VMs mapping with entire scheduling process for 15 tasks of montage workflow

TaskID	*EST*	*EFT*	*ACT*	*VMID*	*CloudID*
T_1	0	13	13	3	2
T_2	0	14	14	1	1
T_3	0	12	12	4	2
T_4	26	39	13	1	1
T_5	13	28	15	4	2
T_6	14	32	18	2	1
T_7	28	41	13	3	2
T_8	13	26	13	3	2
T_9	39	59	20	1	1
T_{10}	59	70	11	1	1
T_{11}	70	92	22	2	1
T_{12}	70	90	20	1	1
T_{13}	79	93	14	4	2
T_{14}	92	108	16	1	1
T_{15}	108	124	16	1	1

The corresponding Normalized $Temp\_ECT_{i,j}$ only for T_2 will be as follows:

$$\begin{matrix} \\ T_2 \end{matrix} \begin{pmatrix} VM_1 & VM_2 & VM_3 & VM_4 \\ 0 & .2143 & .9286 & 1.00 \end{pmatrix}$$

Here, also $Thrsh(d_{T_i}) < MaxN\_Temp\_ECT_{i,j}$ therefore task T_3 is also placed in large batch (b_large). Further by calculating EST and EFT, the task T_2 is mapped on the VM_1 at CS_1.

Now at level-2 in the DAG example, there are 5 tasks T_4, T_5, T_6, T_7 and T_8 which are in the ready list QR_{l_i} and the tasks T_1, T_2 and T_3 have already been mapped on the VM_3, VM_1 and VM_4, respectively.

Therefore, after the scheduling of tasks T_1, T_2 and T_3 the machine available time $Avail_j$ for VM_1, VM_2, VM_3 and VM_4 will be 14, 0, 13 and 12, respectively. But T_4 has the predecessors tasks T_1 and T_2 with the data transfer time $DT_{i,j}$ 13 and 15, respectively. So according to Algorithm 4 from step-5 to step 11, the $Temp\_ECT_{i,j}$ for the tasks T_4, T_5, T_6, T_7 and T_8 and all the 4 VMs VM_1, VM_2, VM_3 and VM_4 will be updated as follows:

Table 5 Gantt chart for NMMWS

	VM_4	0∼12	13∼28	28∼79	79∼93	93∼124		
CS_2		T_2	T_5	*	T_{13}	*		
	VM_3	0∼13	13∼26	28∼41	41∼124			
		T_1	T_8	T_7	*			
	VM_2	0∼14	14∼32	32∼70	70∼92	92∼124		
CS_1		*	T_6	*	T_{11}	*		
	VM_1	0∼14	26∼39	39∼59	59∼70	70∼90	92∼108	108∼124
		T_2	T_4	T_9	T_{10}	T_{12}	T_{14}	T_{15}

Table 6 Gantt chart for Min–Min scheduling

	VM_4	0∼12	13∼28	28∼83	83∼95	93∼140				
CS_2		T_3	T_5	*	T_{11}	*				
	VM_3	0∼13	13∼26	26∼28	28∼41	41∼140				
		T_1	T_8	*	T_7	*				
	VM_2	0∼14	14∼32	32∼70	70∼91	91∼140				
CS_1		*	T_6	*	T_{12}	*				
	VM_1	0∼14	14∼26	26∼39	39∼59	59∼70	70∼86	86∼108	108∼124	124∼140
		T_2	*	T_4	T_9	T_{10}	T_{13}	*	T_{14}	T_{15}

Table 7 Gantt chart for Max–Min scheduling

	VM_4	0∼13	13∼28	28∼77	77∼90	90∼102	102∼147	
CS_2		*	T_5	*	T_{10}	T_{11}	*	
	VM_3	0∼13	13∼51	51∼64	64∼77	77∼99	99∼147	
		T_1	*	T_7	T_9	T_{12}	*	
	VM_2	0∼33	33∼53	53∼147				
CS_1		*	T_8	*				
	VM_1	0∼14	14∼33	33∼46	46∼59	59∼99	99∼115	115∼131	131∼147
		T_2	T_4	T_6	*	T_{13}	T_{14}	T_{15}	*

Table 8 Gantt chart for HEFT scheduling

	VM_4	0∼12	12∼13	13∼28	28∼42	42∼91	91∼105	105∼152		
CS_2		T_3	*	T_5	T_7	*	T_{13}	*		
	VM_3	0∼13	13∼25	25∼38	38∼51	51∼64	64∼152			
		T_1	*	T_6	T_8	T_9	*			
	VM_2	0∼82	82∼103	103∼152						
CS_1		*	T_{12}	*						
	VM_1	0∼14	14∼26	26∼39	39∼71	71∼82	82∼104	104∼120	120∼136	136∼152
		T_2	*	T_4	*	T_{10}	T_{11}	*	T_{14}	T_{15}

$$Temp\_ECT_{i,j} = \begin{matrix} \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \end{matrix} \begin{pmatrix} VM_1 & VM_2 & VM_3 & VM_4 \\ 39 & 46 & 42 & 43 \\ 48 & 49 & 34 & 28 \\ 42 & 32 & 38 & 43 \\ 45 & 45 & 41 & 42 \\ 49 & 51 & 26 & 30 \end{pmatrix}$$

After that, the normalization of $Temp\_ECT_{i,j}$ has been done step by step. Now according to $Thrsh(d_{T_i})$ the tasks are placed into small batch or large batch similar to previous steps for T_1, T_2 and T_3 at level-1. For this, the overall task-VM mapping and scheduling is demonstrated in Table 4.

The Gantt chart for the montage workflow application of Fig. 1 against the *ECT* matrix of Table 3 is shown in Table 5. It is noteworthy that an asterisk ('*') represents the idle time slot of virtual machine and '∼' denotes duration of the assigned task in the Gantt chart.

We also produce the Gantt charts for the existing min–min DAG scheduling [11], max–min DAG scheduling [11] and

Table 9 Makespan and average cloud utilization comparison for NMMWS, Min–Min, Max–Min, DLS and HEFT scheduling algorithm

	NMMWS	Min–Min	Max–Min	DLS	HEFT
Makespan	124	140	147	197	152
Avg. cloud utilization (in %)	46.37	39.821	38.77	31.97	36.18

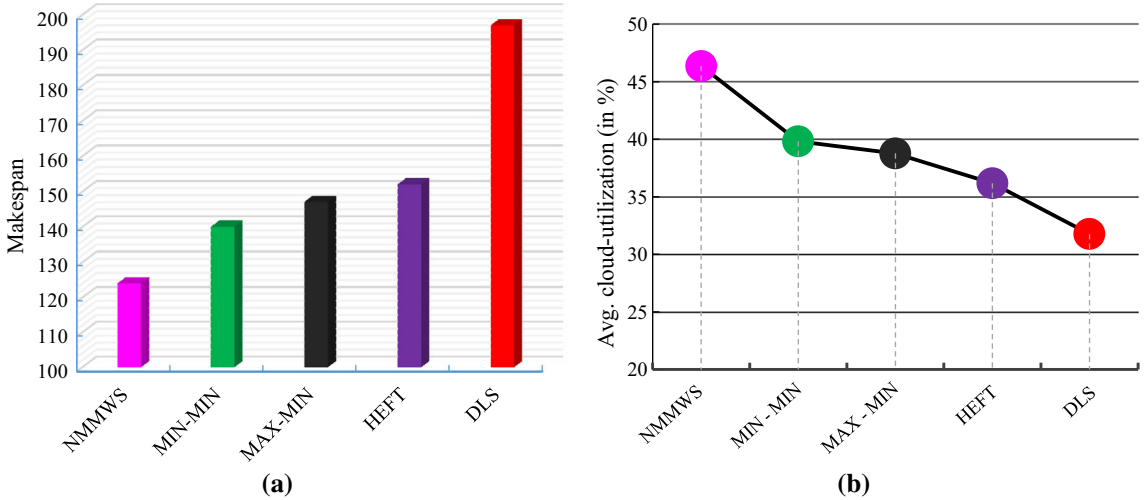

Fig. 2 Makespan and avg. cloud utilization comparison for a given montage workflows. **a** Makespan and **b** avg. cloud utilization

HEFT scheduling [9] algorithm as shown in Tables 6, 7 and 8, respectively.

From the above Gantt charts, we can see that proposed workflow scheduling algorithm NMMWS performs far better than the existing algorithm in terms of cloud makespan and average cloud utilization. It is also well visualized in Table 9 and Fig. 2

5 Experimental Results and Performance Evaluation

The *cloud makespan* and *average cloud utilization* are two metrics which are desirable to evaluate the performance of the proposed algorithm.

5.1 Simulation Setup

We simulated the proposed algorithm for five benchmark workflow applications, namely Cybershake, Epigenomic, Inspiral, Montage and Sipht by generating various CCRs (e.g., 0.05, 0.5, 1, 2 and 4) without violating the core structure of these workflows. Our simulation setup uses MATLAB R2012a running the entire simulation on Microsoft Windows 7 Professional operating system 64-bit with the system configuration as Intel core i5-3230M CPU @ 3.20 GHz. In general, to assess the performance of the simulation results, we arrange every workflow into three different three distinctive sizes. The workflow of 30–50 numbers of tasks is set in small workflow group and the workflows having around 100 tasks is put in medium size work process. Another workflow classification having around 1000 tasks is put in the vast (large) size of the workflow. Aside from this, we additionally produced distinctive value of *CCR* DAGs without changing the core structure of the workflows. The *CCR* is the DAG property by which we can find the nature or type of the workflow. Extremely lower estimated value of the *CCR* is termed as the compute-intensive workflow and higher estimated value of the *CCR* termed as the data-intensive workflow application. So by taking distinctive mixes of *ECT* and *DT*, a matrix is formed according to the uniform as well as random distribution of real data value, we also created a range from lower to higher *CCR* workflows for execution assessment of the proposed algorithm.

5.2 Simulation Run on Benchmark Workflow Applications

To evaluate the proposed scheduling algorithm, we have tested the algorithm on benchmark workflow applications which belong to different scientific domain. These workflows are well described in [1,2] that we used for the simulation of proposed algorithm. The brief description of these workflows is given as follows:

(1) Cybershake: The Cybershake workflows are used to depict earthquake threats in any region by fabricating seismic exposure curve. It is an earthquake deathtraps presentation which is prerequisite high amount of data loading time and I/O operation. Therefore, Cybershake is recognized as the data-intensive workflow.
(2) Epigenomic: This type of workflows record the genomic state of humanoid cell through genome inclusive measurement. Fundamentally, it is a bioinformatics solicitation which is prerequisite a lot of records transmission time along with high computation time. Therefore, Epigenomic workflows categorize in both categories (1) data-intensive and (2) compute-intensive workflow applications.
(3) Inspiral (LIGO): This class of workflow belongs to gravitational physics application for analyzing the gravitational surfs twisted by different event happening in the space. They necessitate huge data storage time as well as computation time. Therefore, Inspiral workflows are classified as the compute-intensive workflow.
(4) Montage: It is a cosmological application which is needed a lot of data transfer time rather than the computation time. Montage workflows are the collection of mosaics data which is obtained from the telescope as astronomical image mosaic engine. Therefore, it also categorized as the data-intensive workflows.
(5) Sipht: This workflow belongs to bioinformatics project at Harvard which systematizes the search for untranslated RNAs (sRNAs) for bacteriological replicators in the NCBI database.

Figure 3 represents a structural overview of all type of workflow applications (from Fig. 3a–e). The exhaustive description of all five workflows is given in [1]. In Fig. 3, we can take a look of the diverse combinations of task nodes and workflow properties (i.e., data distribution, data aggregation and pipelining). Here, for the simulation run we took five types of scientific workflows in which each workflow has of five different *CCR* value (0.05–4.00). These workflows are generated from the DAX (XML file format) using MATLAB R2014a by applying uniform distribution on task nodes and edges weights. These XML files are converted in to a 2-D array of size $[n, m]$ estimated computation (ECT) matrix and upper triangular adjacency square matrix $DT_{i,j}$ holding task dependencies among the tasks. We follow the random increasing order to generate ECT matrix and fixed task dependencies for $DT_{i,j}$ as per the simulation model [2]. The detail descriptions are as follows.

Step 1: We set a wide range and variety of task nodes such as 25–30, 50–60, 100 and 1000 and a set of VMs as 5, 10, 20 and 50 those are deployed on up to 4 different cloud servers. We followed the interval of each ECT instance so that all the tasks of the workflow applications are holding float value in the interval of [907.654, 100,000.123]. **Step 2:** We also generate five different *CCR* value-based dependencies among tasks nodes as $DT_{i,j}$ using uniform distribution of edge weights. **Step 3:** We evaluate workflow scheduling by incorporating compute-intensiveness to data-intensiveness corresponding to the *CCR* value.

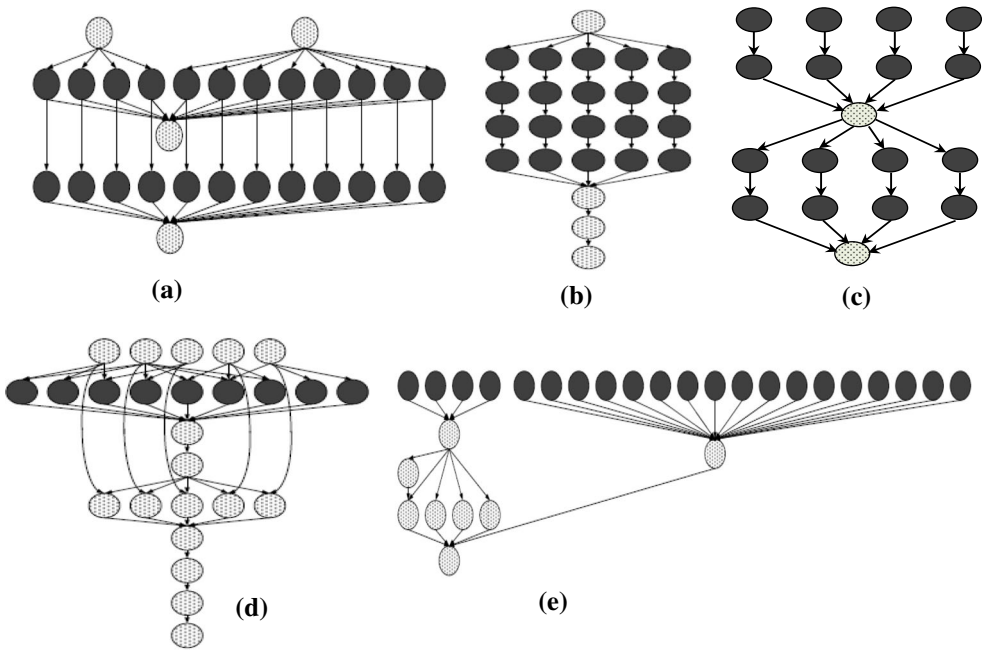

Fig. 3 Structure of all different five workflows from the various scientific domains. **a** Cybershake, **b** Epigenomic, **c** Inspiral, **d** Montage, **e** Sipht

5.2.1 Comparison of Makespan

It is clearly visualized from Fig. 4a–e that NMMWS provides minimum makespan for all small size workflow applications having the range from .05 to 4.0 of CCR value. NMMWS shows comparatively better performance for all the medium and large size workflow applications of huge size, i.e., upto 100–1000 task nodes. It is also clearly shown in Fig. 5a, b. Here, we can observe that NMMWS lags behind min–min, max–min, HEFT and DLS in almost all cases.

5.2.2 Comparison of Average Cloud Utilization

We can observe from Fig. 6 that NMMWS provides better average cloud utilization for every type of workflow applications. The differences are much remarkable, and it shows that the overall performance of NMMWS is far better than DLS, min–min, max–min and HEFT for all the five scientific workflow applications.

5.3 Analysis of Variance (ANOVA) test

Here, we validate the performance of the proposed algorithm through the popular statistical test ANOVA [18] which is used to decide if means of experimental results of various algorithms have any significant statistical differences. To fulfill this purpose, we define a null hypothesis as:

$$H_0: \mu_{\text{NMMWS}} = \mu_{\text{Max–Min}} = \mu_{\text{Min–Min}} = \mu_{\text{HEFT}} = \mu_{\text{DLS}}.$$

Similarly, the alternative hypothesis can be defined as H_1: Means are not equal. Conversely, it is to be assumed that at least one of the means is unlike in substitute hypothesis. We conducted the ANOVA for each five workflows on different CCR value by supposing that alpha is 0.05. The outcomes of ANOVA test are shown in Tables 10, 11, 12, 13 and 14 for Cybershake, Epigenomic, Inspiral, Montage and Sipht workflows, respectively. Note that, we deliberate 20 different instances of each algorithm for the ANOVA test of bench-

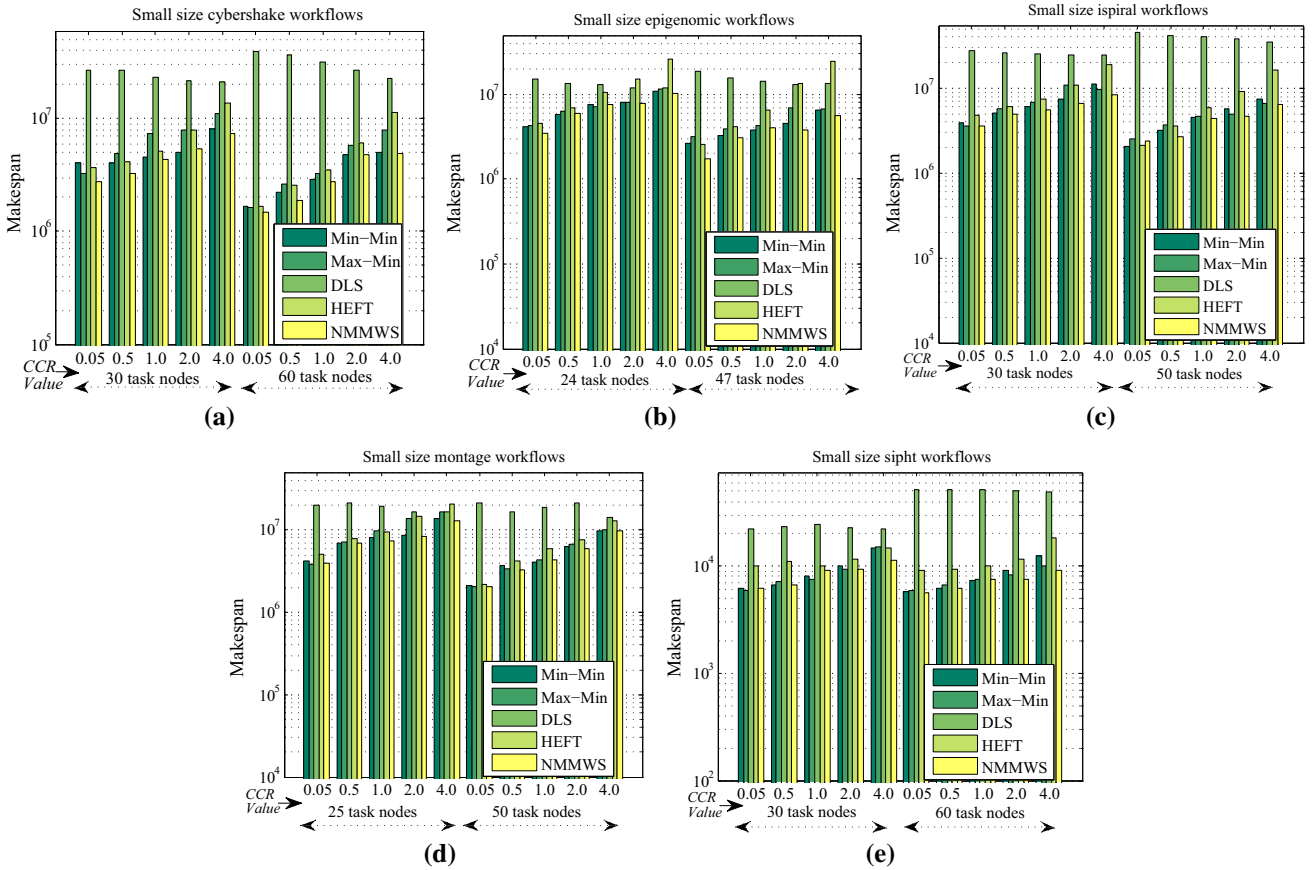

Fig. 4 Comparison of makespan for all small size workflows on different CCR value: **a** Cybershake, **b** Epigenomic, **c** Inspiral, **d** Montage and **d** Sipht workflows

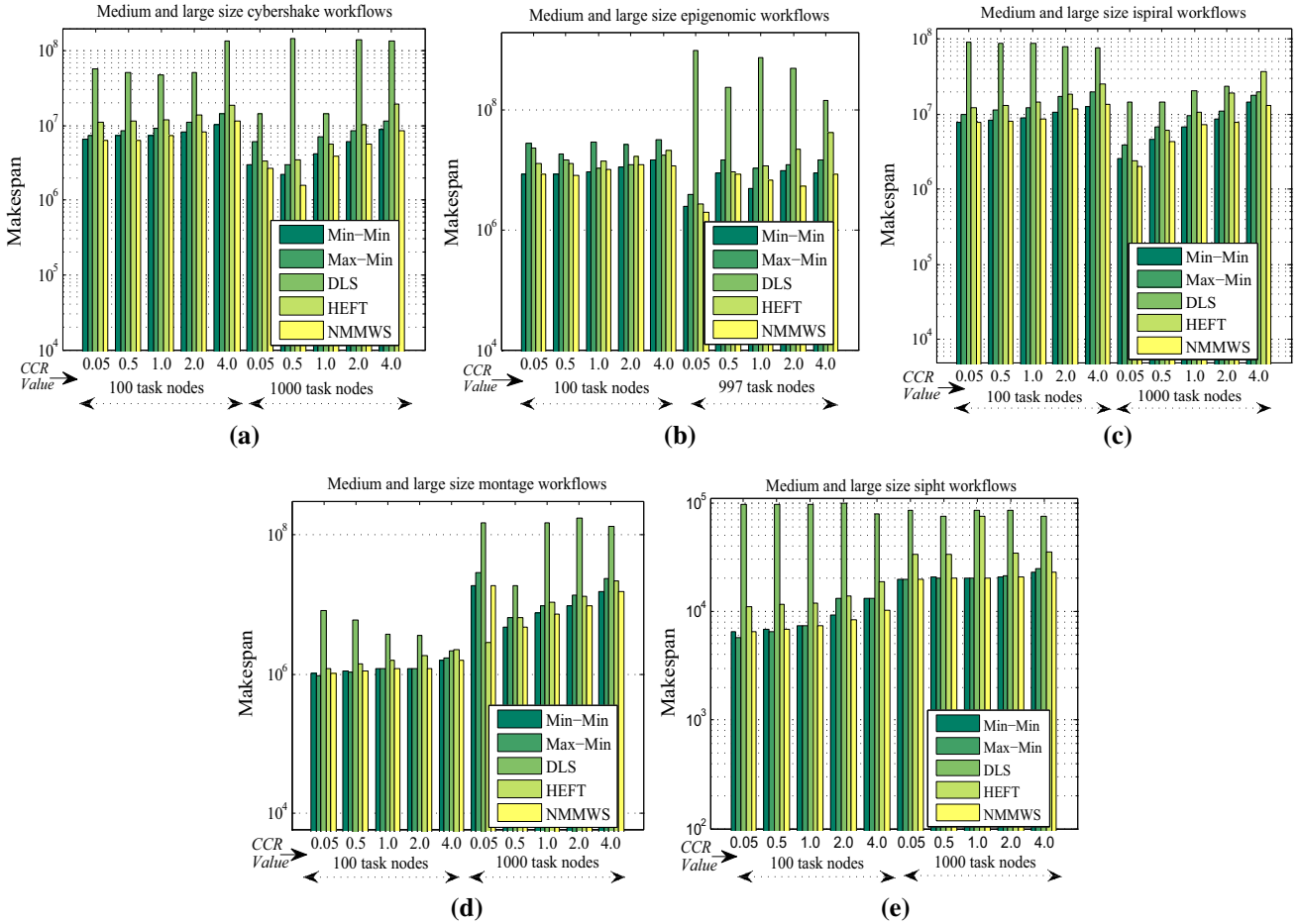

Fig. 5 Comparison of makespan for all medium and large size workflows on different CCR value: **a** Cybershake, **b** Epigenomic, **c** Inspiral, **d** Montage and **d** Sipht Workflows

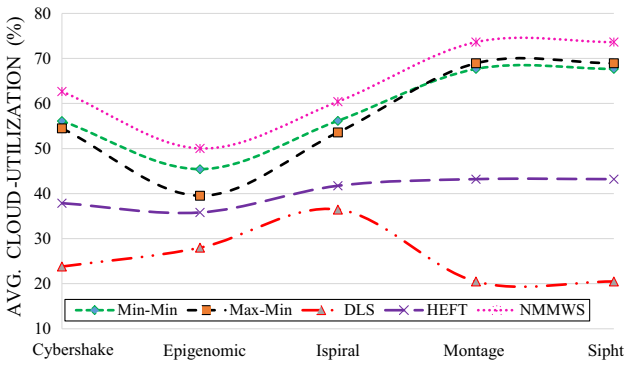

Fig. 6 Avg. cloud utilization comparison for all type workflows

mark workflow applications. As $F_{\text{statistic}} > F_{\text{critical}}$, so we discard the null hypothesis for all the benchmark workflows. It also shows that the means of the all the groups of algorithms are not equal because the P_{value} is much smaller than alpha (value is 0.05). In the tables (from Tables 10, 11, 12, 13, 14) SS refers as the sum of square and SV refers as source of variation, respectively.

6 Conclusion

In this paper, we have presented a dynamic list-based workflow scheduling algorithm. The main objective of the proposed work is to minimize makespan by keeping maximum average cloud utilization. We have shown the simulation results for five benchmark scientific workflow applications on different *CCR* value, and performance evaluation has been done by comparing the experimental results with min–min, max–min, HEFT and DLS as per their adequacy. The comparison results have displayed that the proposed algorithm performs better than these algorithms. By considering variation in the *CCR* value, we have also shown that the proposed algorithm is quite capable to schedule both data-intensive and compute-intensive workflows. Moreover, the experimental results has been validated by the well-known statistical test ANOVA. However, the proposed algorithm lacks in considering the other aspects such failure of VMs, handling multiple workflows and dynamic workflow scheduling. Therefore, the combination of these issues opens a direction of our future

Table 10 ANOVA test results for all size Cybershake workflows

Source of variation	*SS*	*df*	*MS*	F_{value}	P_{value}	$F_{critical}$
Between groups	1.24299e+16	04	3.11e+15	18.27645	5.08e−11	2.472927
Within groups	1.53024e+16	90	1.7e+14			
Total	2.77323e+16	94				

Table 11 ANOVA test results for all size Epigenomic workflows

Source of variation	*SS*	*df*	*MS*	F_{value}	P_{value}	$F_{critical}$
Between groups	2.90674e+17	04	7.26686e+16	4.477618016	0.002403686	2.472927039
Within groups	1.46064e+18	90	1.62293e+16			
Total	1.75131e+18	94				

Table 12 ANOVA test results for all size Inspiral workflows

Source of variation	*SS*	*df*	*MS*	F_{value}	P_{value}	$F_{critical}$
Between groups	2.57081e+18	04	6.42703e+17	7.994636911	1.46312e−05	2.472927039
Within groups	7.23526e+18	90	8.03918e+16			
Total	9.80607e+18	94				

Table 13 ANOVA test results for all size Montage workflows

Source of variation	*SS*	*df*	*MS*	F_{value}	P_{value}	$F_{critical}$
Between groups	2.29744e+18	04	5.74359e+17	6.60216954	0.000104986	2.472927039
Within groups	7.8296e+18	90	8.69955e+16			
Total	1.0127e+19	94				

Table 14 ANOVA test results for all size Sipht workflows

Source of variation	*SS*	*df*	*MS*	F_{value}	P_{value}	$F_{critical}$
Between groups	1.2005e+12	04	3.00125e+11	9.117881244	3.138e−06	2.472927039
Within groups	2.96245e+12	90	32916086927			
Total	4.16295e+12	94				

works. The attempt will also be made to simulate any newly proposed algorithms in the real cloud environment.

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Juve, G.; Chervenak, A.; Deelman, E.; Bharathi, S.; Mehta, G.; Vahi, K.: Characterizing and profiling scientific workflows. Future Gener. Comput. Syst. **29**(3), 682–692 (2013)
2. https://confluence.pegasus.isi.edu/display/pegasus/ Workflow Generator. Accessed on 25 Nov 2016
3. Wieczorek, M.; Prodan, R.; Fahringer, T.: Scheduling of scientific workflows in the ASKALON grid environment. ACM SIGMOD Rec. **34**(3), 56–62 (2005)
4. Cooper, K.; Dasgupta, A.; Kennedy, K.; Koelbel, C.; Mandal, A.; Marin, G.; Mazina, M.; Mellor-Crummey, J.; Berman, F.; Casanova, H.; Chien, A.: New grid scheduling and rescheduling methods in the GrADS project. In: 18th International on Parallel and Distributed Processing Symposium. IEEE (2004)
5. Alkhanak, E.N.; Lee, S.P.; Rezaei, R.; Parizi, R.M.: Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: a review, classifications, and open issues. J. Syst. Softw. **113**, 1–26 (2016)
6. Durillo, J.J.; Prodan, R.; Barbosa, J.G.: Pareto tradeoff scheduling of workflows on federated commercial clouds. Simul. Model. Pract. Theory **58**, 95–111 (2015)
7. Buyya, R.; Vecchiola, C.; Selvi, S.T.: Mastering Cloud Computing: Foundations and Applications Programming. Morgan Kaufmann, Los Altos (2013)

8. Bochenina, K.; Butakov, N.; Boukhanovsky, A.: Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments. Future Gener. Comput. Syst. **55**, 51–61 (2016)
9. Topcuoglu, H.; Hariri, S.; Min-You, W.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**, 260–274 (2002)
10. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
11. Cao, H.; Jin, H.; Wu, X.; Wu, S.; Shi, X.: DAGMap: efficient and dependable scheduling of DAG workflow job in grid. J. Supercomput. **51**(2), 201–223 (2010)
12. Li, J.; Qiu, M.; Ming, Z.; Quan, G.; Qin, X.; Gu, Z.: Online optimization for scheduling preemptable tasks on IaaS cloud system. J. Parallel Distrib. Comput. **72**, 666–677 (2012)
13. Panda, S.K.; Jana, P.K.: Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment. Inf. Syst. Front. (2016). https://doi.org/10.1007/s10796-016-9683-5
14. Panda, S.K.; Jana, P.K.: Uncertainty-based QoS min–min algorithm for heterogeneous multi-cloud environment. Arabian J. Sci. Eng. **41**(8), 3003–3025 (2016)
15. Ding, Y.; Qin, X.; Liu, L.; Wang, T.: Efficient scheduling of virtual machines in cloud with deadline constraint. Future Gener. Comput. Syst. **50**, 62–74 (2015)
16. Braun, T.D.; Siegel, H.J.; Beck, N.; Boloni, L.L.; Maheswaran, M.; Reuther, A.I.; Robertson, J.P.; Theys, M.D.; Yao, B.; Hensgen, D.; Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
17. Panda, S.K.; Jana, P.K.: Efficient task scheduling algorithms for heterogeneous multi-cloud environment. J. Supercomput. **71**(4), 1505–1533 (2015)
18. Muller, K.E.; Fetterman, B.A.: Regression and ANOVA: An Integrated Approach Using SAS Software. SAS Publisher Cary (2002)
19. Vasile, M.; Pop, F.; Tutueanu, R.; Cristea, V.; Kolodziej, J.: Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. Future Gener. Comput. Syst. **51**, 61–71 (2015)
20. Celaya, J.; Arronategui, U.: Fair scheduling of bag-of-tasks applications on large-scale platforms. Future Gener. Comput. Syst. **49**, 28–44 (2015)
21. Mao, M.; Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of International Conference for High Performance Computing Networking, Storage and Analysis. ACM (2011)
22. Gorbenko, A.; Popov, V.: Task-resource scheduling problem. Int. J. Autom. Comput. **9**, 429–441 (2012)
23. Panda, S.K.; Jana, P.K.: A multi-objective task scheduling algorithm for heterogeneous multi-cloud environment. In: International Conference on Electronic Design, Computer Networks and Automated Verification, pp. 82–87. IEEE (2015)
24. Gupta, I.; Kumar, M.S.; Jana, P.K.: Compute-intensive workflow scheduling in multi-cloud environment. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 315–321. IEEE (2016)
25. Ming, G.; Li, H.: An improved algorithm based on max–min for cloud task scheduling. In: Recent Advances in Computer Science and Information Engineering, Lecture Notes in Electrical Engineering, vol. 125, pp. 217–223 (2012)
26. Ibarra, O.H.; Kim, C.E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. ACM (JACM) **24**(2), 280–289 (1977)
27. Malawski, M.; Juve, G.; Deelman, E.; Nabrzyski, J.: Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. Future Gener. Comput. Syst. **48**, 1–8 (2015)
28. Liu, Y.; Zhang, C.; Li, B.; Niu, J.: DeMS: a hybrid scheme of task scheduling and load balancing in computer clusters. J. Netw. Comput. Appl. **83**, 213–220 (2015)
29. Ergu, D.; Kou, G.; Peng, Y.; Shi, Y.; Shi, Y.: The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment. J. Supercomput. **64**, 835–848 (2013)
30. OpenNebula, http://archives.opennebula.org/documentation:rel4.4:schg. Accessed on 16 July 2016
31. Rodriguez, M.A.; Buyya, R.: Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**, 222–235 (2014)
32. Yu, B.; Yuan, X.; Wang, J.: Short-term hydro-thermal scheduling using particle swarm optimization method. Energy Convers. Manag. **48**(7), 1902–1908 (2007)
33. Rashvand, H.F.; Salah, K.; Calero, J.M.A.; Harn, L.: Distributed security for multi-agent systems review and applications. IET Inf. Secur. **4**(4), 188–201 (2010)
34. www.nimbusproject.org/docs/2.5/changelog.html. Accessed on 15 July 2016
35. Freund, R.F.; Gherrity, M.; Ambrosius, S.; Campbell, M.; Halderman, M.; Hensgen, D.; Keith, E.; Kidd, T.; Kussow, M.; Lima, J.D.; Mirabile, F.; Moore, L.; Rust, B.; Siegel, H.J.: Scheduling resources in multi-user. In: Heterogeneous, Computing Environments with SmartNet, 7th IEEE Heterogeneous Computing Workshop, pp. 184–199 (1998) Comput. Mach. **24**(2), 280–289 (1977)
36. Braun, F.N.: https://code.google.com/p/hcspchc/source/browse/trunk/AE/ProblemInstns/HCSP. Accessed on 15 May 2016
37. Salah, K.; Elbadawi, K.; Boutabaa, R.: An analytical model for estimating cloud resources of elastic services. J. Netw. Syst. Manag. **24**(2), 285–308 (2016)
38. Nudd, G.R.; Kerbyson, D.J.; Papaefstathiou, E.; Perry, S.C.; Harper, J.S.; Wilcox, D.V.: PACEA toolset for the performance prediction of parallel and distributed systems. Int. J. High Perform. Comput. Appl. **14**(3), 228–251 (2000)