J Supercomput (2018) 74:370–385
https://doi.org/10.1007/s11227-017-2133-4

An adaptive task allocation technique for green cloud computing

Sambit Kumar Mishra[1] · Deepak Puthal[2] · Bibhudatta Sahoo[1] · Sajay Kumar Jena[1] · Mohammad S. Obaidat[3,4]

Published online: 2 September 2017

Abstract The rapid growth of todays IT demands reflects the increased use of cloud data centers. Reducing computational power consumption in cloud data center is one of the challenging research issues in the current era. Power consumption is directly proportional to a number of resources assigned to tasks. So, the power consumption can be reduced by a demotivating number of resources assigned to serve the task. In this paper, we have studied the energy consumption in cloud environment based on varieties of services and achieved the provisions to promote green cloud computing. This will help to preserve overall energy consumption of the system. Task allocation in the cloud computing environment is a well-known problem, and through this problem, we can facilitate green cloud computing. We have proposed an adaptive task allocation algorithm for the heterogeneous cloud environment. We applied the proposed technique to minimize the makespan of the cloud system and reduce the energy consumption. We have evaluated the proposed algorithm in CloudSim simulation environment, and simulation results show that our proposed algorithm is energy efficient in cloud environment compared to other existing techniques.

✉ Deepak Puthal
dputhal88@gmail.com

Sambit Kumar Mishra
skmishra.nitrkl@gmail.com

Mohammad S. Obaidat
msobaidat@gmail.com

1 National Institute of Technology, Rourkela, India

2 University of Technology Sydney, Sydney, Australia

3 Fordham University, Bronx, NY, USA

4 University of Jordan, Amman, Jordan

Keywords Cloud computing · Energy consumption · Makespan · Task allocation · Virtual machine

1 Introduction

Cloud computing has been developed as one of the creative platforms which give dependable, virtualized and adaptable cloud resources over the Internet. To utilize these resources, the cloud customers do not require any hardware or software infrastructure. They can lease the cloud resources on demand from any place on the planet just by spending for that utilized resources like electricity supply. The cloud user got these resources from the cloud service providers (CSPs) through laptops, PCs, mobile devices, etc. The CSP offers membership to the clients for different services like infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS).

Scheduling or allocation is the procedure of choosing how to submit resources to a variety of conceivable tasks. This allocation of resources is a basic and critical job for any business perspectives. The resource allocation in the cloud environment is carried out by the CSP. If the cloud resources are scheduled properly, then the effectiveness and performance of the system will be enhanced [1,2]. Therefore, resource scheduling strikes a significant role in a system for getting successful and productive results. It increases throughput and also balances the computing load to avoid overloading and underloading [3]. This scheduling or allocation problem is a well-known NP-complete problem [4]. Therefore, suboptimal solutions are proposed for the allocation problem. Various researchers work on this scheduling problem by considering different objectives like energy consumption, resource utilization, makespan, load balancing, guaranteeing Quality of Service (QoS), enhancing throughput and Service Level Agreement (SLA) completion [1–3,5,6]. Here, in this work, makespan minimization is taken as primary objective along with the energy consumption of the system. Three crucial factors affect the task allocation in the cloud environments [7]. Those are waiting time in the task queue, execution time, and the relative performance of the off-premise cloud compared to the on-premise machines. The users could achieve the optimal turnaround time if they know the waiting time in the task queue, and the execution time in the cloud environment. However, this knowledge for the user is not available in advance. The problem of this work can be expressed as follows. Given a set of tasks, a set of virtual machines (VMs), and an arrangement of evaluations values in the form of an expected time to compute (ETC) matrix to demonstrate how well every VM can perform each task, such that the aggregate evaluations values are maximized [8,19]. An ETC matrix has the expected execution time of all task one by one in different VMs. This ETC matrix introduces the heterogeneity of tasks as well as the resources (i.e., VMs). A single task can be executed in different VMs with different execution times which represent machine heterogeneity. Similarly, a single VM takes a different amount of time to execute different tasks that represent task heterogeneity.

Energy-efficient data centers address green cloud computing system. Green cloud is a new terminology in the computing world in which consolidation of user requests

or cloud resources plays a significant role. The green cloud system has various distinct components such as the energy consumed for computation, communication and the physical infrastructure of a data center. To optimize the energy consumption of a cloud data center, either one can go for the allocation of user requests (tasks) to the existing finite set of VMs or can for the distribution of cloud resources (VMs) to a finite set of physical hosts. In this work, the mapping of a batch of tasks to VMs is performed. Among the three service models, the IaaS has a tendency to give a more fertile ground to reduce makespan as well as energy by task allocation.

Definition 1 (*Makespan*) Makespan is the maximum time required to complete a finite number of tasks by a VM among all virtual machines after allocation of tasks to VMs.

Definition 2 (*Server availability*) A physical server is available when the server has sufficient computing resources to host the required number of virtual machines.

Contributions: This work has the following key contributions:

- It presented a system model that includes a host model, virtual machine model and task model.
- It presented an adaptive allocation of tasks to virtual machines that adjust the execution time of tasks dynamically.
- It evaluated the effect of different scenarios for makespan and energy consumption of the cloud system.
- It provided a comparative analysis among our algorithm, baseline (random) algorithm and round Robin algorithm.

The remaining work is organized as follows: Sect. 2 discusses literature that focused on this problem and its solution. Section 3 discusses the problem statement along with some assumptions required for the system. The next section describes system model including task and resource model. Section 5 presents the proposed algorithm followed by a hand-tracing example. In Sect. 6, the evaluation of the algorithm is illustrated. Finally, we conclude our work with the future scope.

2 Related work

Research in the area of cloud computing attracted greater attention in the recent times due to the huge capabilities in the IT field. Task allocation in the cloud environment is one of the important research problems in order to optimize time, energy, cost, etc. [9]. In [1], Rimal and Maier have proposed an approach for the scheduling the workflow to minimize makespan, the task execution costs, and to use the idle cloud resources effectively. They have only considered the CPU-intensive task and dealt with both structured and unstructured workflow scheduling. They have compared their approach with the FCFS, EAST (Extensible Argonne Scheduling System) backfilling and minimum completion time scheduling policies.

Several states correspond to the different energy consumptions of the CPU, main memory and secondary storage. From the recent work, it is found that most processors support running state, idle state, sleep state and off state. In [12], Mills-Tettey et al. have

explained the allocation problem with changing costs using bipartite graph where the edge cost varies. For the solution of their problem, they have used a dynamic Hungarian algorithm, and they have presented correctness proofs of the algorithm including its efficiency. In [13], a near-optimal solution of task assignment problem is proposed to maximize the cumulative profit or to reduce the energy cost of the cloud data center. They have compared their work with the random algorithm in the cloud environment. In [14], Penner et al. have presented an algorithm for task distribution that dynamically adjusts the costs based on the previous allocation. The goal is to provide load balancing and collocating task executions.

Beloglazov et al. [2] proposed an architectural framework and principles for green cloud computing. Their method includes architectural principles for energy-efficient management of clouds and energy-efficient resource allocation policies. In their work, the authors have validated their approach by a performance evaluation study using the CloudSim toolkit. A collaboration of optimization scheduling and estimation techniques with the power consumption in a cloud environment is shown in [3]. This technique improves the performance for green cloud computing. A randomized algorithm is proposed for task allocation [15] by reducing both time and space complexities. A middleware is proposed for performing a hybrid simulation of large-scale critical systems [16]. This middleware allows a multi-objective optimization approach to optimize the simulation task allocation on a private cloud.

To improve the performance of the cloud system and also to conserve energy consumption of the data centers, we propose an algorithm for the allocation of cloud tasks to the cloud resources. We found that the resource allocation using the round Robin and baseline (random allocation) algorithms is well accepted by a large number of researchers [20–23]. Hence, we have compared the performance of our proposed technique with these standard algorithms using simulation analysis. The random-based task allocation algorithm assigns tasks to VMs on a random basis without any constraint.

3 Problem statement

The assignment problem of a huge number of tasks to a finite number of VMs in the cloud environment is addressed as task allocation problem. There are n number of tasks $\{T_1, T_2, \ldots, T_n\}$ and m number of VMs in the cloud system. The assignment of these n tasks to m VMs should be efficient so that the makespan, as well as the energy consumption of the system, is optimized. Makespan of the system is calculated from the Execution Time of Virtual machine (ETV) set.

$$\text{Makespan}\,(M) = M\text{ax}(\text{ETV}_i) \tag{1}$$

Here ETV_i represents the execution time of ith virtual machine. Our objective is to minimize M as in Eq. (1). Another important performance parameter is energy consumption of the cloud system to execute a finite number of heterogeneous tasks.

The energy consumption of the system is calculated by adding the energy consumption of individual VMs using Eq. (2).

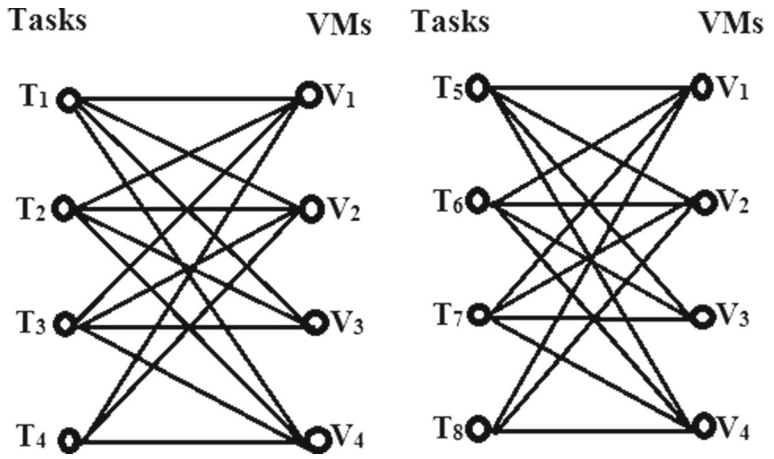

Fig. 1 Two bipartite graphs with eight tasks and four VMs

$$\text{Energy consumption } (E) = \sum_{i=1}^{m} E_i \tag{2}$$

Here E_i is the energy consumption of ith VM, $1 \leq i \leq m$. A virtual machine can be in one state, i.e., active or idle. So, to calculate the energy consumption of a VM, both active and idle state energy consumptions are added. The idle state time of a VM is calculated by subtracting the active state time from the makespan of the system. A_i Joules/Million Instruction (J/MI) is the energy consumption of ith VM in an active state, and I_i J/MI is in an idle state. The energy consumption of a VM in idle state (I_i) is 60% of A_i [5]. The energy consumption of a virtual machine mainly depends on the speed (Million Instruction Per Second, MIPS) of the VM as in Eq. (3) adopted from [10,11] and (4).

$$A_i = 10^{-8} \times (\text{MIPS}_i)^2 \text{ J/MI} \tag{3}$$

$$I_i = 0.6 \times A_i \text{ J/MI} \tag{4}$$

Energy consumption of all VMs is required to calculate the total energy consumption of the system (E) by using Eq. (5).

$$E_i = \text{ETV}_i \times A_i + (M - \text{ETV}_i) \times I_i \tag{5}$$

Let say $G = \{T, V, E\}$ be a bipartite graph as shown in Fig. 1, T the set of tasks, V the set of VMs and E the set of edges that represent the allocation.

It assumes a matrix edge cost, ET C_{ij}, $0 \leq i \leq n$, $0 \leq j \leq m$, where n is the total number of tasks and m is the total number of VMs. Missing edge has a higher ETC value, i.e., infinity as illustrated in Table 1. If a task cannot be assigned to a VM, then we make the corresponding ETC value as infinity (∞).

Table 1 ETC matrix of eight tasks and four VMs

ETC_{ij}	V1	V2	V3	V4
T1	7	9	6	9
T2	9	8	5	5
T3	4	3	7	6
T4	6	6	∞	7
T5	4	5	2	3
T6	8	9	5	7
T7	15	14	14	15
T8	5	7	∞	8

3.1 Assumptions

Virtualization technology supports the creation of multiple VMs on any of the possible host. The workload submitted to the cloud is considered to be in the form of tasks. The scheduler allocates the tasks to VMs of different hosts for execution. In order to solve the task allocation problem in the cloud environment, we have considered the following assumptions.

- Each task is assigned to a single virtual machine.
- Tasks cannot be preempted once they begin to execute on a VM.
- When a VM executes a task, there are no priority distributions between the tasks with the VM.
- A VM cannot remain idle when the tasks are in the waiting queue of the VM.
- Each VM has a single core.
- There is no failure of VM during execution of tasks.
- The tasks are independent in nature.

4 System model

In this section, we present a scheduling model for a cloud data center along with the task model and VM model in a cloud data center. The scheduling model of a cloud data center where the CSP schedules the task of users to the cloud resources is shown in Fig. 2. Here, we have an assumption that the cloud system has enough resources to provide services to the cloud user. The cloud users ($User_1, User_2, \ldots, User_n$) submit their tasks to the CSP, and all tasks ($T_1, T_2, \ldots, T_n$) are in a task queue. The task classifier classifies all tasks into a CPU-bound task set, urgent CPU-bound task set, IO-bound task set, urgent IO-bound task set, and all classified tasks are submitted to the respective task queue. CPU-bound tasks and urgent CPU-bound tasks are submitted to the SCHEDULER1, and the tasks are allocated among x number of CPU-bound virtual machines ($CV_1, CV_2, \ldots, CV_x$). Similarly, all IO-bound tasks and urgent IO-bound tasks are submitted to the SCHEDULER2, and the tasks are allocated among y number of IO-bound virtual machines ($IV_1, IV_2, \ldots, IV_y$). Both schedulers schedule the urgent tasks first, and if the urgent tasks queue is empty, then they assign regular CPU-bound and IO-bound tasks to the respective VMs.

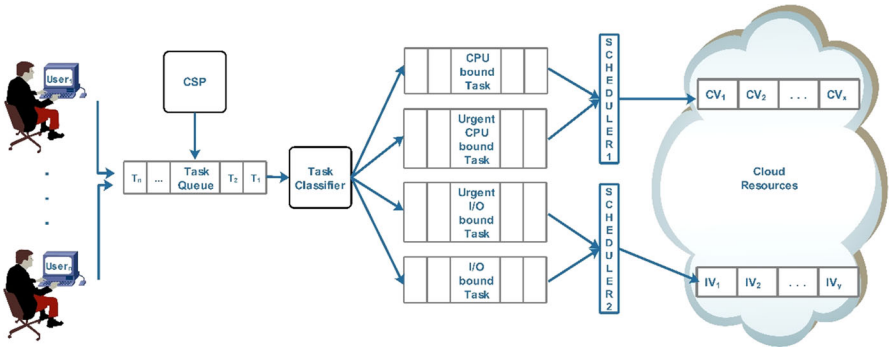

Fig. 2 An adaptive scheduling model for cloud system

4.1 Task model

In the cloud environment, a large number of cloud users submit their independent tasks to the cloud service provider and access services from the cloud without understanding the system infrastructure. The tasks are heterogeneous in terms of length of the tasks and resource requirement of the tasks. There are n (finite) number of tasks, and the set is $T = \{T_1, T_2, \ldots, T_n\}$. Each task $T_i, 1 \leq i \leq n$ has five tuples, i.e., $T_i = \{W_i, \text{CPU}_i, M_i, \lambda_i, R_i\}$, where W_i is the workload of service T_i in terms of MI, CPU_i is the CPU time required for the service T_i, M_i is the main memory requirement for the service T_i, λ_i is the bandwidth requirement of service T_i and R_i represents the task type. R_i value is 0 if T_i CPU-intensive, 1 if T_i urgent CPU-intensive, 2 if T_i IO-intensive and 3 if T_i urgent IO-intensive.

4.1.1 Resource model

A cloud system can be developed from a single data center or from multiple data centers according to the resource requirement. To consider a general cloud system model, the data center set D has p number of data centers, i.e., $D = \{D_1, D_2, \ldots, D_p\}$. Each data center has numerous hosts (and the set of hosts on ith data center is $H_i = \{H_{i1}, H_{i2}, \ldots, H_{i|Hi|}\}$) in the cloud environment. Each host $H_{ij}, 1 \leq i \leq p, 1 \leq j \leq |H|$ can be modeled as six tuples, i.e., $H_{ij} = \{\text{PE}_{ij}, S_{ij}, M_{ij}, \text{SS}_{ij}, \lambda_{ij}, \text{VMM}_{ij}\}$, where

- H_{ij} represents jth host of ith data center.
- PE_{ij} is the number of processing elements or cores of H_{ij}.
- S_{ij} is the processing speed of H_{ij} in terms of MIPS.
- M_{ij} is the host main memory size of H_{ij}.
- SS_{ij} is the secondary memory size of H_{ij}.
- λ_{ij} is the total bandwidth provided to H_{ij}.
- VMM_{ij} is the Virtual Machine Manager (VMM) running on the host H_{ij}.

Algorithm 1 Adaptive Task Allocation Algorithm (*ATAA*)
Input: *ETC* matrix, DAG graphs.
Output: *ET*: Execution time of all VMs.

```
1.  B = Batch Size of Tasks (= number of VMs).
2.  N_T = Total number of tasks.
3.  ET = 0
4.  for i = 1 to N_T/B do
5.      BETC = [(i − 1) × B + 1] to [i × B] rows of ETC
6.      for j = 1 to B do
7.          for k = 1 to B do
8.              METC[k, j] = BETC[k, j] + ET[j]
9.          end for
10.     end for
11.     R_METC = RowUpdate(METC).
12.     C_METC = ColumnUpdate(R_METC).
13.     AMETC = Assigned_METC(C_METC).
14.     if each row of AMETC has an assigned 0 then
15.         Allocate tasks to the corresponding VM.
16.         ET = ET + METC value of the allocation.
17.         Continue for the next (i + 1)^th iteration.
18.     else
19.         C_METC = Update_METC(C_METC)
20.         Goto Step 13.
21.     end if
22. end for
```

One of the major advantages of cloud system is virtualization of cloud resources. This virtualization mechanism provides flexibility of partitioning the resources of the host into various virtual machines. A VMM, which is running on a host, is mainly responsible for the maintenance of all VMs on that host. Each host H_{ij} has finite number of virtual machines (and the set of VMs on jth host of ith data center is $V_{ij} = \left\{V^1_{ij,} V^2_{ij,} \ldots, V^k_{ij}\right\}$). Each VM has five tuples, i.e., $V^k_{ij} = \left\{\mathrm{PE}^k_{ij,} S^k_{ij,} M^k_{ij,} \mathrm{SS}^k_{ij,} \lambda^k_{ij}\right\}$.

- V^k_{ij} represents kth VM running on jth host of ith data center.
- PE^k_{ij} is the number of processing elements or cores of V^k_{ij}.
- S^k_{ij} is the processing speed of V^k_{ij} in terms of MIPS.
- M^k_{ij} is the main memory size of V^k_{ij}.
- SS^k_{ij} is the secondary memory size of V^k_{ij}.
- λ^k_{ij} is the total bandwidth provided to V^k_{ij}.

5 Proposed task allocation algorithm

This section discusses the details of the proposed scheduling algorithm, and the purpose is to reduce the overall energy consumption by minimizing the makespan of the system. For Algorithm 1: Adaptive Task Allocation Algorithm (*ATAA*), the ETC matrix of all tasks of a queue is provided as input. There are four queues for the task because of four type of tasks as shown in Fig. 2. Therefore, the *ATAA* algorithm is applied for

all four-task queues. The ETC matrix has the completion time of tasks on all virtual machines according to the type of tasks. The output of *ATAA* algorithm is *ET* vector, which has time required by all VMs. Initially, the time required by all VMs, *ET* is set to zero. The tasks are allocated to the cloud resources (VMs) batch-wise. Batch size of the tasks is same as the total number of VMs. To allocate one batch of tasks, a portion of *ETC* matrix is required, i.e., *BETC*. This *BETC* is modified by adding the previous execution time of corresponding VM and forms the modified *ETC* (*METC*) matrix. In *RowUpdate*, the smallest element of each row is subtracted from the elements of that row. In *ColumnUpdate*, the smallest element of each column is subtracted from the elements of that column. Then, the *Assigned METC* subroutine is called after which the allocation of tasks is determined by using the marking procedure.

Algorithm 2 *Assigned_METC*(*C_METC*)
Input: *C_METC*.
Output: *AMETC*: After assignment on *C_METC*.

1. **for** each row **do**
2. **if** a single unmarked 0 is there **then**
3. Mark the 0 as assigned.
4. Ignore the elements of the corresponding row and column of the assigned element in step 3.
5. **end if**
6. **end for**
7. **for** each column **do**
8. **if** a single unmarked 0 is there **then**
9. Mark the 0 assigned.
10. Ignore the elements of the corresponding row and column of the assigned element in step 9.
11. **end if**
12. **end for**
13. Return *AMETC*.

The marking will be done for the elements row-wise first and if required then go for column-wise. If a single 0 is present in a row, then mark that 0 as gray color and strike all 0's across the corresponding column. Similarly, for each column, we follow the same marking procedure as performed for the row. If the allocation is successful, then the *ET* vector is updated according to the allocation of a batch of tasks. If the allocation is not possible (or step 14 of *ATAA* algorithm is not satisfied), then *Update METC* subroutine is called which updates the *C_METC* matrix by using ticking procedure. Here, we tick all unassigned rows (i.e., the row with no marked zeros). If the ticked rows have a 0, then tick the corresponding columns, and if the ticked column has an assignment (i.e., marked zeros), then tick the corresponding rows. This ticking procedure will repeatedly be continued until no more ticking is possible. After that, lines are to be drawn through unticked rows and ticked columns. The smallest number (θ) of the matrix that has no lines passing through is subtracted from the elements of the matrix that have one line passing through and add the θ value to the matrix elements that have two lines passing through. The updated *METC* matrix is returned to Algorithm 1, where again Algorithm 2 is called for the marking procedure. Then, again step 4: *Assigned_METC* subroutine is called and repeat this procedure till we have a successful allocation.

If the total number of tasks is n and the total number of VMs is m, then the loop started in step 4 of Algorithm 1 will run for $n = m$ times. Their inner loops, as well as subroutines, will run at most for $O(m^2)$ times. Therefore, the time complexity of the allocation result after performing the *ATAA* algorithm is $O(mn)$, $m \ll n$. Because the time complexity of Algorithm 2 and Algorithm 3 is $O(m^2)$ which are called from Algorithm 1. Therefore, the time complexity of *ATAA* algorithm is $O((n/m) \times (m^2 + m^2 + \ldots + k \text{ times})) = O(mn)$, $where\ k$ is an integer constant. The time complexity of the basic Hungarian algorithm for solving assignment problem is $O(n^3)$ [12], which is larger than the time complexity of *ATAA*, i.e., $O(n^2) > O(mn)$.

Algorithm 3 *Update_METC*
Input: *C_METC*.
Output: *C_METC*: Updated matrix.

1. Tick all unassigned rows.
2. **if** Ticked row has a 0 **then**
3. Tick the corresponding column.
4. **end if**
5. **if** Ticked column has an assignment **then**
6. Tick the corresponding row.
7. **end if**
8. Repeat steps 2 to 7 till no more ticking is possible.
9. Draw lines through unticked rows and ticked columns.
10. θ = Smallest number that have no lines passing through.
11. $C\_METC_{ij} = C\_METC_{ij} - \theta$, If no lines passing through.
12. $C\_METC_{ij} = C\_METC_{ij}$, If one line passing through.
13. $C\_METC_{ij} = C\_METC_{ij} + \theta$, If two lines passing through.
14. Return the updated *METC*.

5.1 Example

The explanation of the example will carry from the ETC matrix as shown in Table 1. There are eight tasks, and we have to allocate those tasks to four VMs so that the makespan is minimum. For this example, we consider four as the batch size of the tasks, which is same as the number of VMs. So, Table 1 is split into Tables 2 and 4. The step-by-step solution of the allocation problem for the given example is explained below with the corresponding remark (Table 3).

Table 4 has four tasks (T5 to T8) of Table 1 and is represented as *BETC*. Before allocation of tasks to the cloud resources (VMs), the addition of previous execution times of all VMs will be done and the updated matrix is shown in the second row

Table 2 ETC matrix of four tasks and four VMs

ETC_{ij}	V1	V2	V3	V4
T1	7	9	6	9
T2	9	8	5	5
T3	4	3	7	6
T4	6	6	∞	7

Table 3 Description of allocation problem for Table 2 data

Stages	Remarks
7 9 6 9 9 8 5 5 4 3 7 6 6 6 ∞ 7	This is the ETC matrix for first four tasks ($T1$ to $T4$). Initially, the execution time of each VM is 0. So, no updating is required
1 3 0 3 4 3 0 0 1 0 4 3 0 0 ∞ 1	After RowUpdate step. Here, the smallest element of each row is subtracted from all elements of that row
1 3 0 3 4 3 0 0 1 0 4 3 0 0 ∞ 1	After ColumnUpdate step. Here, the smallest element of each column is subtracted from all elements of that column
1 3 0 3 4 3 θ 0 1 0 4 3 0 θ ∞ 1	After applying Algorithm 2. For each row, mark 0 as gray color, if that row has a single unmarked 0. Then, for each column, mark 0 as a gray color, if that column has a single unmarked 0. Make strike to all 0's which is located on assigned rows of columns
T1 T2 T3 T4 V3 V4 V2 V1	After successful allocation. The execution time of each VM is *ET* & the vector is as follows 6 5 3 6

Table 4 ETC matrix of four tasks and four VMs

ETC_{ij}	V1	V2	V3	V4
T5	4	5	2	3
T6	8	9	5	7
T7	15	14	14	15
T8	5	7	∞	8

of Table 5. The *RowUpdate* and *ColumnUpdate* methods are performed (Algorithm 1). As *C_METC* [1,3] contains a single unmarked 0, that particular cell is colored as gray, and the striking operation is done on all the 0's in the corresponding rows and columns. The marking operation is accomplished using column value also. In this case, we can mark *C_METC* [1,4] and colored it as gray. Still, this algorithm (Algorithm 2) is not able to mark 0's for each row and column. Therefore, we follow another procedure as stated in Algorithm 3. Primarily, tick all unassigned rows (rows 2 and 3) and the corresponding column (column 3) where 0 is present. If there is a marked 0 present in that particular column, then tick that row too (row 1). The next step of the algorithm allows crossing a line between the unticked rows and ticked columns. As shown in Table 5 (seventh row), we have colored the lines as light gray. Select the smallest number (θ) in the matrix that has one line passing through it. Here, the smallest number is 1. Then, subtract the θ from the elements of the matrix that have no lines passing through it and add the particular to those two lines passing through it. After doing this, the updated matrix shown in Table 5 (eighth row) is obtained.

Table 5 Description of allocation problem for Table 4 data

Stages	Remarks
4 5 2 3 8 9 5 7 15 14 14 15 5 7 ∞ 8	This is the ETC matrix for first four tasks (*T*5 to *T*8)
10 10 5 9 14 14 8 13 21 19 17 21 11 12 ∞ 14	After adding the execution time of each VM i.e., in *ET* vector to the current ETC matrix
5 5 0 4 6 6 0 5 4 2 0 4 0 1 ∞ 3	After RowUpdate step. Here, the smallest element of each row is subtracted from all elements of that row
5 4 0 1 6 5 0 2 4 1 0 1 0 0 ∞ 0	After ColumnUpdate step. Here, the smallest element of each column is subtracted from all elements of that column
5 4 0 1 6 5 θ 2 4 1 θ 1 0 θ ∞ θ	After Algorithm 2. For each row, mark 0 as a gray color, if that row has a single unmarked 0. Then, for each column, mark 0 as gray color, if that column has a single unmarked 0. Make strike to all 0's that are located on assigned rows of columns. Still, 2 rows are unassigned
✓ 5 4 0 1 ✓ 6 5 0 2 ✓ 4 1 0 1 ✓ 0 0 ∞ 0	After applying steps 1-8 of Algorithm 3. Make a tick mark on all unassigned rows and for those rows where ever 0 found, make a tick mark to the corresponding columns.
✓ 5 4 0 1 ✓ 6 5 0 2 ✓ 4 1 0 1 ✓ 0 0 ∞ 0	Step 9 of Algorithm 3. Make lines on the unticked rows and also make lines on the ticked columns
4 3 0 0 5 4 0 1 3 0 0 0 0 0 ∞ 0	After applying steps 10-14 of Algorithm 3. Make lines on the unticked rows and also make lines on the ticked columns. Here, instead of lines, we color those rows and columns with light gray color
4 3 θ 0 5 4 0 1 3 0 θ θ 0 θ ∞ θ	After applying Algorithm 2 again. For each row, mark 0 as gray color, if that row has a single unmarked 0. Then, for each column, mark 0 as a gray color, if that column has a single unmarked 0. Make strike to all 0's that are located on assigned rows of columns
T1 T2 T3 T4 V V V V 4 3 2 1	After successful allocation. The execution time of each VM is *ET* & the vector is as follows 9 8 19 11

Then, apply Algorithm 2 to this updated matrix as shown in the ninth row of Table 5. Then, we get the successful allocation result along with the execution time of each VM.

6 Evaluation of the algorithm

The CloudSim 3.0.3 simulator is used to simulate the service allocation problem using the *ATAA* algorithm. The CloudSim simulator provides a generalized simulation framework for modeling purpose, simulation purpose and experimenting purpose in the cloud infrastructure [17,18]. Let TH and MH represent task heterogeneity and machine heterogeneity, respectively. In this paper, we have adopted the range-based ETC generation algorithm [19]. The ETC matrix for simulation is formed by employing two uniform distributions $U(1, \text{TH})$ and $U(1, \text{MH})$ and are realized as:$1 + (\text{TH} - 1) \times \text{rand}(1)$ and $1 + (\text{MH} - 1) \times \text{rand}(1)$, where rand() function generates a value within (0, 1). For the ETC matrix generation, we have used $\text{TH} = 1000$ and $\text{MH} = 50$, respectively. We have compared the proposed algorithm with the baseline algorithm (where the allocation performed randomly, which means any task can be allocated to a VM with $1 = m$ probability) and with the Round-Robin algorithm (which is the default scheduling algorithm in the CloudSim simulator). We have considered makespan and energy consumption of the system as performance metrics to evaluate the algorithm.

For the simulation purpose, we have considered a single data center with multiple heterogeneous hosts and 'Xen' as VMM. We have estimated tasks and resources with diverse requirements as the environment support for heterogeneous system modeling. The key properties of the elements for the simulation are:

- Speed of VM (MIPS) : 200, 400, 500, 800, 1000, 2000, 4000, 5000, 8000, 10,000;
- Main memory size (RAM) : 512, 1024, 2048, 4096, 8192, 16,384;
- We also varied the secondary storage size;

We have examined the performance of the cloud system in two different scenarios as follows.

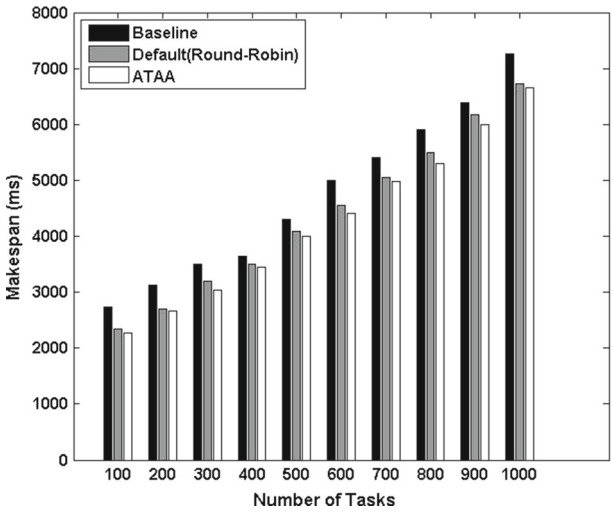

Fig. 3 Makespan comparison as shown in Scenario 1

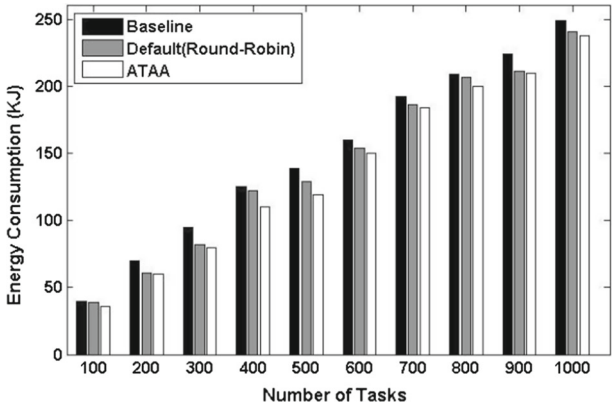

Fig. 4 Energy consumption comparison as shown in Scenario 1

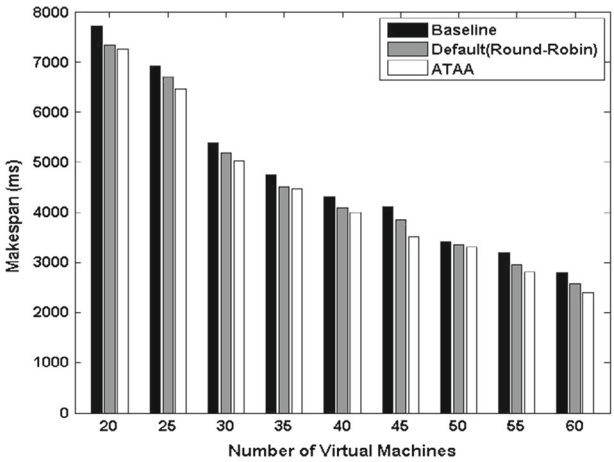

Fig. 5 Makespan comparison as shown in Scenario 2

Scenario 1 In this scenario, the number of virtual machines is fixed, i.e., 40, and the number of tasks varies from 100 to 1000 in the interval of 100. The makespan of the system is determined for each set of tasks (for all three algorithms), and the energy consumption is calculated. The comparison graph for this scenario is shown in Fig. 3 (makespan), and Fig. 4 (energy consumption).

Scenario 2 In this scenario, the number of tasks is fixed, i.e., 500, and the number of VMs varies from 20 to 60 in the interval of 5. Here, also the makespan and energy consumption are calculated. The comparison graph for this scenario is shown in Fig. 5 (makespan) and Fig. 6 (energy consumption).

The makespan of the system increases when the number of input tasks increases for a fixed number of VMs as shown in Fig. 3. The makespan value is more when the number of VMs is less as displayed in Fig. 5. This makespan value is gradually

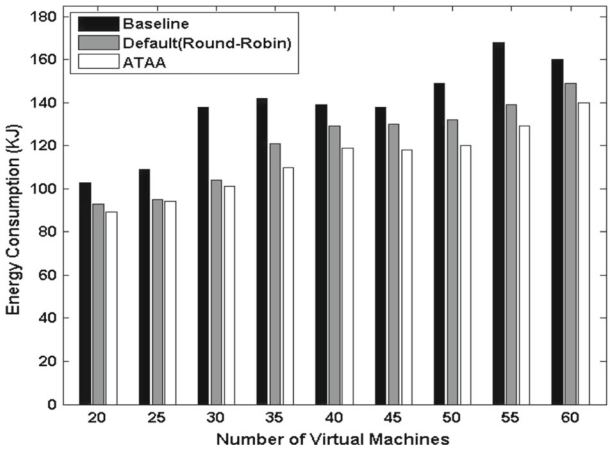

Fig. 6 Energy consumption comparison as shown in Scenario 2

reduced as the number of VMs increases as presented in the bar chart (Fig. 5). The energy consumption of the system increases rapidly when the number of tasks increases with a certain number of VMs as shown in Fig. 6. However, the energy consumption of the system increases slowly when the number of VMs increases as shown in Fig. 6. Here, the energy consumption rate is slow due to fixed number of tasks.

7 Conclusion

This paper incorporates a detailed evaluation of the system framework toward the resource distribution and task allocation in the cloud computing system. In this paper, we have proposed a novel task-scheduling algorithm *ATAA* in the cloud environment. We presented a system model including task model and resource model (includes host model, VM model) for the cloud environment. Due to the importance of urgent CPU-bound tasks and urgent IO-bound tasks for the proposed method, the percentage of execution of tasks before the deadline (i.e., one of the SLA constraints) is much more, i.e., a higher rate of execution achieved. The work in this paper also addresses the heterogeneity with the help of ETC matrix. A hand-traced example of the algorithm for given ETC is discussed which shows the basic steps of the algorithm. The principle of mathematical modeling is taken into consideration to evaluate the performance of the proposed allocation technique. The simulation results for the comparison of algorithms result in favor of *ATAA* algorithm. This finding can further lead to the dynamic form of resource allocation algorithms that permit preempted tasks according to a supplied priority.

References

1. Rimal BP, Maier M (2017) Workflow Scheduling in Multi-Tenant Cloud Computing Environments. IEEE Trans Parallel Distrib Syst 28(1):290–304

2. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Fut Gen Comput Syst 28(5):755–768
3. Vakilinia S, Heidarpour B, Cheriet M (2016) Energy efficient resource allocation in cloud computing environments. IEEE Access 4:8544–8557
4. Gary MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. WH Freemann, New York
5. Sampaio AM, Barbosa JG, Prodan R (2015) PIASA: a power and interference aware resource management strategy for heterogeneous workloads in cloud data centers. Simul Model Pract Theory 57:142–160
6. Ali HGEDH, Saroit IA, Kotb AM (2017) Grouped tasks scheduling algorithm based on QoS in cloud computing network. Egypt Inform J 18(1):11–19
7. Cunha RL, Rodrigues ER, Tizzei LP, Netto MA (2017) Job placement advisor based on turnaround predictions for HPC hybrid clouds. Fut Gen Comput Syst 67:35–46
8. Munkres J (1957) Algorithms for the assignment and transportation problems. J Soc Ind Appl Math 5(1):32–38
9. Thaman J, Singh M (2017) Green cloud environment by using robust planning algorithm. Egypt Inform J. doi:10.1016/j.eij.2017.02.001
10. Grochowski E, Annavaram M (2006) Energy per instruction trends in Intel microprocessors. Technol Intel Mag 4(3):1–8
11. Shi T, Yang M, Li X, Lei Q, Jiang Y (2016) An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds. Pervasive Mob Comput 27:90–105
12. Mills-Tettey GA, Stentz A, Dias MB (2007) The dynamic Hungarian algorithm for the assignment problem with changing costs. Technical report Carnegie Mellon University (CMU-RI-TR-07-27)
13. Wang Y, Chen S, Pedram M (2013) Service level agreement-based joint application environment assignment and resource allocation in cloud computing systems. In: Green Technologies Conference, IEEE, Apr 2013, pp 167–174
14. Penner T, Johnson A, Van Slyke B, Guirguis M, Gu Q (2014) Transient clouds: assignment and collaborative execution of tasks on mobile devices. In: Global Communications Conference (GLOBECOM), IEEE, Dec 2014, pp 2801–2806
15. Kentros S, Kari C, Kiayias A, Russell A (2015) Asynchronous adaptive task allocation. In: 35th International Conference on Distributed Computing Systems (ICDCS), 2015 IEEE, June 2015, pp 83–92
16. Ficco M, Di Martino B, Pietrantuono R, Russo S (2017) Optimized task allocation on private cloud for hybrid simulation of large-scale critical systems. Fut Gen Comput Syst 74:104–118
17. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp 41(1):23–50
18. Buyya R, Ranjan R, Calheiros RN (2009) Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: challenges and opportunities. In: International Conference on High Performance Computing and Simulation, 2009. HPCS'09, IEEE, June 2009, pp 1–11
19. Ali S, Siegel HJ, Maheswaran M, Hensgen D (2000). Task execution time modeling for heterogeneous computing systems. In: 9th Proceedings on Heterogeneous Computing Workshop (HCW'2000), IEEE, pp 185–199
20. Dong Z, Liu N, Rojas-Cessa R (2015) Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers. J Cloud Comput 4(1):5
21. Chen J, Li K, Tang Z, Bilal K, Yu S, Weng C, Li K (2017) A parallel random forest algorithm for big data in a Spark cloud computing environment. IEEE Trans Parallel Distrib Syst 28(4):919–933
22. Devi DC, Uthariaraj VR (2016). Load balancing in cloud computing environment using Improved Weighted Round Robin Algorithm for nonpreemptive dependent tasks. Sci World J 2016:1–14. doi:10.1155/2016/3896065
23. Elmougy S, Sarhan S, Joundy M (2017) A novel hybrid of shortest job first and round Robin with dynamic variable quantum time task scheduling technique. J Cloud Comput 6(1):12