Cluster Comput (2018) 21:163–176
https://doi.org/10.1007/s10586-017-0976-3

ScHeduling of jobs and Adaptive Resource Provisioning (SHARP) approach in cloud computing

Dinesh Komarasamy[1] · **Vijayalakshmi Muthuswamy**[1]

Received: 11 February 2017 / Revised: 17 May 2017 / Accepted: 5 June 2017 / Published online: 20 June 2017

Abstract Affordability of appropriate computing resources for satisfying prerequisites of Service Level Agreement (SLA) of clients and optimal utilization of cloud service providers are limited in the present scenario of cloud computing. To overcome these limitations, researchers have exploited various scheduling algorithms to process the deadline based autonomous jobs. The scheduling algorithms however do not support multiprocessor demand and adaptive resource provisioning. This inference triggers to propose a new approach called ScHeduling of jobs and Adaptive Resource Provisioning (SHARP) in cloud computing to handle independent jobs that processes the jobs in a multilevel manner. The SHARP approach embeds multiple criteria decision analysis to preprocess the jobs, multiple attribute job scheduling to prioritize the jobs and adaptive resource provisioning to provide resources dynamically. These contributions alleviate SLA violations in terms of deadline, upgrade client satisfaction and enhance resource utilization. The empirical studies verify the proposed approach in a cloud environment and show the necessity of the proposed approach to support elastic resource provisioning and meet SLA requirements.

Keywords Cloud computing · Job scheduling · Backfilling algorithm · Elastic resource provisioning · Resource utilization

✉ Dinesh Komarasamy
dinesh.komarasamy@gmail.com; dinesh.nova@gmail.com

Vijayalakshmi Muthuswamy
vijim@annauniv.edu

1 Department of Information Science and Technology, College of Engineering, Anna University, Chennai 600025, India

1 Introduction

With the development of advanced Internet technology, the vast infrastructure is framed as a virtualized cloud infrastructure by interconnecting the resources scattered globally [1–3]. The computing capacity of the virtualized cloud infrastructure has been partitioned into many virtual machines with the aid of virtualization technique [4]. The computing capacity of the Virtual Machines (VMs) rely on the data center policy or Cloud Service Providers (CSPs) which may either be homogeneous where the VMs have equal processing speed or heterogeneous where the VMs have different processing speed [5]. The CSPs have the ability to afford services to several number of clients simultaneously with the support of VMs in cloud infrastructure [6]. E-commerce, e-learning and e-governance applications of the clients that demand a certain amount of computing power to perform their computation is termed as job [7]. The jobs are categorized into dependent and independent jobs based on their requests and behavior [8]. The dependent jobs need the output of other jobs to complete their execution, whereas autonomous jobs do not require them [9].

At any arbitrary time immense number of end-users request services from cloud. During these situations, the jobs have to be scheduled optimally to meet the Service Level Agreement (SLA), which is an agreement made between the service provider and naïve users before offering a service [10]. Thereby, a job scheduler should play a major role in fulfilling the user requirements, in assisting for optimal utilization of the service provider and also in enhancing the throughput of the VM [11]. To achieve these goals, analysts have proposed several job scheduling algorithms in cloud computing. Among these, most of the researchers have focused on scheduling the deadline based independent jobs that require only uniprocessors for execution [12]. Unfortunately, most of the complex applications of the clients demand multiprocessors for processing their jobs.

This demand leads to the following challenges, (1) the jobs submitted by the clients are blindly acknowledged by the traditional approach which in turn affects the Quality of Service (QoS) of service providers, (2) the existing schedulers are designed focused only on the client-side, (3) even though there is a demand for uniprocessor as well as multiprocessor by the jobs, most of the existing scheduling algorithms are implicitly designed for processing uniprocessor jobs only, (4) the existing algorithms do not provide resources based on the system load and availability of resources, (5) the traditional myths lack in the provisioning of resources dynamically to meet the SLA requirements of the naïve users and service providers.

To meet the above mentioned challenges, this work presents ScHeduling of jobs and Adaptive Resource Provisioning (SHARP) approach in cloud computing for handling the ever changing client's demand. The proposed work encompasses Multiple Criteria Decision Analysis (MCDA), Multiple Attribute Job Scheduling (MAJS) and Adaptive Resource Provisioning (ARP) to process the jobs in a multi-phased manner. Initially, the jobs are preprocessed with the aid of MCDA to avoid malware and unfeasible jobs so as to enhance QoS of the service provider. After which, the jobs are dynamically prioritized using MAJS that relies on the attributes of the jobs and service providers. The prioritized jobs are then mapped to the resources with the aid of ARP, which supports elastic provisioning of resources to fulfill the naïve users' requirements and stabilizes system load. The following section briefly describes the related works. Section 3 describes the entire design of the SHARP approach. Section 4 explains the experimental results and performance study of the proposed work. Section 5 outlines the conclusion and future work.

2 Literature review

This section describes the overview of the job scheduling algorithms that optimally schedule the jobs and resource provisioning algorithms that map the jobs with the appropriate resources at runtime.

2.1 Scheduling algorithms

Job scheduling is a NP-hard optimization problem that needs to process and complete the jobs with the aid of available resources in the cloud [13]. Considering that there are numerous VMs in cloud computing, a couple of naïve users publish their requests with distinct goals. Here, a job scheduler plays a significant role to minimize the makespan of the job by accurately binding the jobs with the available VMs [14,15]. Many researchers have offered algorithms for scheduling the jobs with common objectives such as minimizing the waiting time, minimizing the execution time and cost of execution. With the aid of traditional job scheduling techniques, the jobs are categorized into two types, as dependent jobs and independent jobs. Initially, the number of jobs submitted by the naïve users were lower than the available number of VMs in the resource pool. In such a scenario, the incoming jobs were simply scheduled using First Come First Serve (FCFS) algorithm [16].

The jobs submitted by the naïve users contain both deadline and non-deadline based jobs which are sometimes dense and sparse. Many researchers have introduced several algorithms to complete the jobs within their deadline. Among these, the advanced reservation technique was proposed to schedule the deadline based jobs which reserves the resources for a particular period to complete the jobs within their deadline [17]. Sometimes, the reserved VMs may be idle. But it cannot be utilized by other jobs which are ready for execution as it may affect the jobs which reserved those VMs already. To solve these issues, backfilling algorithm was introduced to utilize the idle VMs and allow jobs that are capable of completing within the idle time period, without affecting the reserved jobs [18]. Further, the deadline based jobs were optimally processed using the Earliest Deadline First (EDF) algorithm which gives high preferences to the job having the earliest deadline to attain the user satisfaction. Since, the EDF algorithm does not support preemption of jobs, the running jobs cannot be preempted when a higher priority job arrives [19]. So, the fully preemptive-EDF (fp-EDF) algorithm was introduced that preempts the running jobs whenever a higher priority job arrives [20]. But it preempts the running jobs without considering the present state of running jobs which in turn degrades the performance of the system. So, the controlled preemptive-EDF (cp-EDF) algorithm was proposed which outperforms EDF algorithm and fp-EDF by preempting the jobs based on expected completion time of running jobs [21]. Later, Global EDF (GEDF) algorithm was proposed which assigns a global deadline to schedule and complete the jobs within their deadline [22]. Further, MapReduce Task Scheduler for Deadline (MTSD) algorithm was proposed to meet the time constraint of the jobs in a heterogeneous environment [23].

The Partial Critical Paths (PCP) algorithm was proposed to schedule the deadline based jobs depending on their path [24]. Later, Infrastructure Cloud PCP (IC-PCP) was presented which has two phases like planning phase and distribution phase. It shares the sub deadline among the nodes for completing the jobs within their deadline [25]. Still the jobs were not able to complete within their deadline, and so the PCP was modified as Enhanced Ic-Pcp with Replication (EIPR) that replicates job and simultaneously run the jobs in several VMs and attempts to complete the jobs within their deadline [26]. And also, a one-tier VM architectural design was adopted to implement the above mentioned algorithms in which VMs process only one job at a time. The jobs were processed with the aid of VMs, but the processing speed of the VMs were not fully utilized. As a result, Adaptive Multilevel Scheduling System (AMSS) was presented which schedules and processes jobs with the support of VMs in a two-tier architecture to optimally utilize the processing speed of the VMs [27]. The two-tier VM architecture encompasses a foreground VM and background VM, which dynamically shares the processing speed of the VM among both VMs [28]. Moreover, a hybrid task scheduling and load balancing algorithm was introduced to schedule and balance the load among the resources in the service provider [29].

2.2 Resource provisioning algorithm

Similar to job scheduling, resource provisioning plays a vital role in completing jobs within their deadline by mapping them with the appropriate resources. From the state of the art, resource provisioning in cloud computing are categorized into several types such as cost aware resource scheduling, efficiency aware resource scheduling, energy-aware resource scheduling, load balancing-aware resource scheduling, QoS-aware resource scheduling and utilization-aware resource scheduling [30]. Among these, Heterogeneous Earliest Finish Time (HEFT) algorithm was proposed to improve the QoS by giving high preference to the job having the earliest finish time among the jobs in a queue. In HEFT algorithm, the jobs are mapped with the appropriate VMs in the resource pool [31]. Further, the Hierarchical Load Balancing Algorithm (HLBA) was proposed to balance the system load. In HLBA algorithm, the jobs were scheduled to resources based on three parameters namely network utilization, memory and idle computing power [32]. Further, the Graphical Load Balance (GLB) algorithm was introduced to define a set of decisions for mapping the jobs with the resources in the cloud environment [33]. Several optimization algorithms such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) were introduced that optimally map the jobs with the resources at runtime [34,35]. Moreover, the genetic algorithm for workload scheduling in cloud based e-learning was introduced to optimize the scheduling of e-learning workloads with a predefined set of conditions [36]. Further, Dynamic Resource Provisioning and Monitoring (DRPM) was proposed to select VMs for running the jobs that minimizes the completion time of the jobs. DRPM embeds host fault detection algorithm that finds and maps the appropriate VMs for running the jobs to minimize the cost [37].

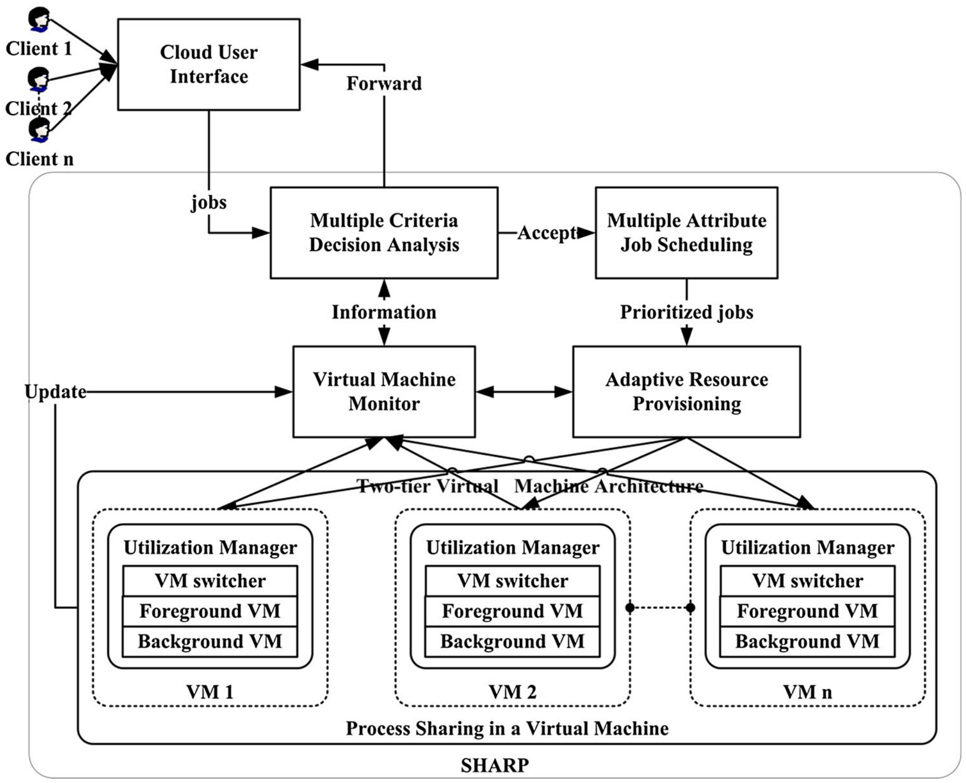
Existing algorithms are focused only to fulfill the client side requirements and concentrate on optimal scheduling of jobs requesting for uniprocessor only. Moreover, the current techniques focus on job scheduling and resource provisioning independently. And also, the existing resource provisioning techniques do not support elastic provisioning of resources. From the state of the art, it is very clear that the existing algorithms do not focus on multiprocessor job and adaptive resource provisioning. Thus, the proposed SHARP approach optimally schedules the incoming jobs to fulfill the user requirements, minimizes the turnaround time, mitigates the number of VMs, minimizes SLA violation and boosts the resource utilization.

3 System framework and problem formulation

Due to the advancement of network technologies, a significant number of naïve users approach cloud computing that has several CSPs $CC = \{CSP_1, CSP_2, ..., CSP_n\}$ to process their jobs in a cost-effective manner. This work is centered on processing jobs efficiently in a CSP. Sometimes, VMs in a CSP are not able to process certain jobs and therefore in turn may affect subsequent jobs, thus leading to SLA violations. Consequently, these jobs ought to be filtered to minimize SLA violations. Furthermore, it is found that the waiting time of the jobs can be decreased by altering the order of their execution. Hence, several existing scheduling algorithms have either statically or dynamically prioritized the jobs that request for a uniprocessor for execution, based on their attributes. After prioritization, the jobs are processed using the appropriate VM present in the CSP. But, an efficient scheduling algorithm has to prioritize the jobs that may either require uniprocessor or multiprocessor for their execution.

Hence, this work proposes a new approach called ScHeduling of jobs and Adaptive Resource Provisioning (SHARP) in cloud computing that meets the requirements. At any given arbitrary time, a naïve user submits a large number of jobs to a cloud system. Among these, some of the jobs may not be processed with the aid of VMs in a CSP. Thus, these jobs need to be preprocessed and forwarded to other CSPs in a cloud system to minimize their SLA violations and to enhance user satisfaction. The jobs may require either uniprocessor or multiprocessor for their execution. The proposed SHARP approach focuses on dynamically prioritizing the jobs based on their attributes and VM attributes. It also maps the jobs with the suitable number of VMs in the CSP depending on the requirements and system load for processing them. During the period of job execution, a job may not use the entire processing speed of the VM and so the VMs can be deployed in a two-tier VM architecture. This additionally helps for dynamic provisioning of VMs based on the need of the jobs and availability of the VMs. SHARP approach also supports scaling of VMs to avoid overloading. The contributions of the proposed SHARP approach are scheduling deadline based independent jobs and provisioning resources

Fig. 1 SHARP architecture



adaptively so that the performance of the CSP and utilization of the resource in CSP are strengthened.

4 Design of SHARP approach

Because of pay per usage model and cost-effective services, the naïve users decide to process their jobs in CSPs rather than processing them in different service vendors. The naïve users could submit various jobs with different demands that could be treated optimally in the cloud system with the aid of the proposed SHARP approach. Figure 1 describes the SHARP architecture. In the proposed SHARP approach, the incoming jobs are gathered by a Cloud User Interface (CUI) that stores the jobs in a job queue J_{queue}. The jobs in J_{queue} are then preprocessed by the MCDA methodology using the various attributes of the jobs and VMs. After preprocessing, the jobs are prioritized using the MAJS algorithm by considering the multiple attributes of jobs along with VMs. After prioritization, the jobs are mapped with the best suitable VMs in the Virtual Machine Monitor (VMM) by the adaptive resource provisioning (ARP) method. After mapping, the jobs have to be processed with the respective VMs. ARP method also supports elastic resource provisioning to stabilize the system load. During execution, a job may not utilize the entire processing speed of a VM and so the VMs need to be deployed in a two-tier VM architecture to utilize the idle processing speed of the VMs. The computing capacity of the VM in a two-tier architecture is partitioned into foreground and background VM to share the processing speed of the VM dynamically. The utilization manager has a detailed information about jobs running in the VMs and the VM switcher allocates the jobs to the foreground and background VM depending upon the priority of the jobs.

A set of instructions needed for processing the requests of naïve users is defined as a job. A job may be either compute-intensive or data-intensive. A job that needs more computation time than data transfer time is termed as compute-intensive job [38]. The proposed work considers only compute-intensive jobs. A job is represented as 'j'. The various tuples of the job are described as given in Eq. (1).

$$j = \left(j^{type}, ls, \vec{T}, \vec{R}\right); \quad \forall j \; in \; J_{queue} \tag{1}$$

where j^{type} represents a job type as either compute-intensive job or data-intensive job, ls represents the length of the job in terms of Million Instructions (MIs) [39,40], $\vec{T}$ contains the detailed time information of the job and $\vec{R}$ contains the information of resource requirements. $\vec{T} = \left\{t^a, t^d\right\}$ where t^a denotes the arrival time of the job and t^d denotes the deadline of the job. A job may request for a multiprocessor for execution that supports concurrent processing of jobs. $\vec{R} = \left\{r^{\min}, r^{\max}\right\}$ where $r^{\min}$ denotes the minimum number of VMs required to process the jobs and $r^{\max}$ represents the maximum number of VMs needed to process the jobs. The VMM stores the information of the VMs in the cloud system and the tuples of the VMM are given in Eq. (2).

$$VMM = \left(VM_a, VM_{busy}, \vec{M}\right) \tag{2}$$

where VM_a is a resource pool that holds the set of available VMs and VM_{busy} represents the set of non-idle VMs in the resource pool. $\vec{M}$ holds the information of the VMs and the tuples of the VMs are described in Eq. (3).

$$\vec{M} = (ps, st)\,; \quad \forall VM\ in\ VMM \tag{3}$$

where ps represents the processing speed of a VM and st represents the storage capacity of a VM. The jobs are initially passed to the MCDA.

4.1 Multiple criteria decision analysis (MCDA)

Multiple criteria decision analysis preprocesses the jobs to identify undesirable jobs or malware jobs so as to minimize the delay in the processing of subsequent jobs. The jobs are preprocessed using the various tuples of the jobs and resources in a twofold criteria. As a first step, the maximum required processing speed of a job is represented as $S_{\max}$ and computed as given in Eq. (4).

$$j_i\left(S_{\max}\right) = \frac{j_i\,(ls)}{j_i\left(r^{\min} * \left(t^d - t^a\right)\right)}; \quad where\ i \in (1, n) \tag{4}$$

where 'n' denotes the total number of jobs submitted to the cloud system during the time interval and stored in J_{queue}. The first criteria of the MCDA is represented as J_{f-MCDA} and computed as given in Eq. (5).

$$J_{f-MCDA} = \begin{cases} 1; & if\ \left(j_i\left(r^{\min}\right) \le |\left(j_i\left(S_{\max}\right) \le VM_k\,(ps)\right)| \le |VM_a|\right) \\ 0; & otherwise \end{cases}\ ; \quad where\ i\ \in (1, n)\,, k \in (1, y) \tag{5}$$

where 'y' denotes the number of VMs in VM_a. Among 'n' jobs, the jobs which satisfy the criteria as defined in Eq. (5) are stored in the J_{f-q} queue. The rejected jobs are stored in J_{r-q} and passed to the second level of preprocessing. In the second step, the minimum processing speed required for a job is represented as $S_{\min}$ and is computed with the aid of various tuples of the job as expressed in Eq. (6).

$$j_x\left(S_{\min}\right) = \frac{j_x\,(ls)}{j_x\left(r^{\max} * \left(t^d - t^a\right)\right)}; \quad \forall j\ in\ J_{r-q} \tag{6}$$

The second criteria of the MCDA is represented as J_{s-MCDA} and computed as given in Eq. (7).

$$J_{s-MCDA} = \begin{cases} 1; & if\ \left(j_x\left(r^{\max}\right) \le |\left(j_x\left(S_{\min}\right) \le VM_k\,(ps)\right)| \le |VM_a|\right) \\ 0; & otherwise \end{cases}\ ; \quad where\ \forall j\ in\ J_{r-q}, k \in (1, y) \tag{7}$$

The accepted jobs of J_{s-MCDA} are appended with the jobs in J_{f-q} and the rejected jobs are forwarded to other CSPs for execution. After preprocessing, the jobs in J_{f-q} are given as input to the Multiple Attribute Job Scheduling phase.

4.2 Multiple attribute job scheduling (MAJS)

The MAJS prioritizes the jobs with the aid of the different attributes of the jobs and VMs. The priority value of 'j' is represented as 'P' and it is computed as given in Eq. (8).

$$P_j = \frac{j\,(ls)/j\,(d)}{VM_{\bar{x}}\,(ps)}; \quad \forall j\ in\ J_{f-q} \tag{8}$$

where $VM_{\bar{x}}$ represents the average processing speed of the VMs in VMM. After computing the priorities of jobs, the jobs in J_{f-q} are sorted in the descending order to give preference to jobs that require more processing speed. After prioritization, the jobs with the same priority are resorted based on the $r^{\min}$ value of 'j'. The algorithm below describes the way how jobs with equal priority are prioritized.

Algorithm

```
(1) Get jobs 'j' from $J_{f-q}$
(2) Compute $P_j$ using equation (8)
(3) Sort $\forall j \; in \; J_{f-q}$ in descending order
(4) for i=2 to n do \\ $\forall j \; in \; J_{f-q}$
(5)     while $P_k = P_i \; ; \forall j \; in \; J_{f-q} \quad where \; k \in (i-1,1)$ do
(6)         $J_{f-q} = \begin{cases} swap(j_i, j_k) & ; if \; j_i(r^{\min}) > j_k(r^{\min}) \; where \; k \in (i-1,0) \\ retain & ; otherwise \end{cases}$
(7)     End while
(8) End for
```

After prioritizing the jobs, the jobs are forwarded to Adaptive Resource Provisioning.

4.3 Adaptive resource provisioning (ARP)

In adaptive resource provisioning (ARP), the system load is initially computed to effectively schedule and complete the jobs within their deadline. The system load is defined as a ratio between the estimated processing speed of the jobs to the processing speed of the available VMs in CSP. The system load of a CSP is represented as ρ and computed as given in Eq. (9).

$$\rho = \frac{\sum_{j in J_{f-q}} j\left(S_{\max}\right) * j\left(r^{\min}\right) + \sum_{j in J_{s-q}} j\left(S_{\max}\right) * j\left(r^{\min}\right) + \sum_{\substack{VM \in VM_{busy}, \\ j_{VM}\left(t^f\right) < \bar{t}^d}} VM(ps)}{\sum_{VM \in VM_a} VM(ps) + \sum_{\substack{VM \in VM_{busy}, \\ j_{VM}\left(t^f\right) < \bar{t}^d}} VM(ps)} \tag{9}$$

$$VM_{req}(ps) = \max \; j\left(S_{\max}\right); \quad \forall j \; in \; J_{f-q} \tag{10}$$

The computing capacity of a CSP is scaled up by creating new VMs only when the VMs present in the CSP are incapable of processing the incoming jobs. The additional processing speed required for processing the jobs in J_{f-q} is represented as A_{ps} and it is computed as given in Eq. (11).

$$A_{ps} = \begin{cases} (\rho - 1) * \left(\sum_{VM \in VM_a} VM(ps) + \sum_{\substack{VM \in VM_{busy}, \\ j_{VM}\left(t^f\right) < \bar{t}^d}} VM(ps) \right); & if \;\; \rho > 1 \\ Retain; & otherwise \end{cases} \tag{11}$$

where t^f denotes the finishing time of the running job, $\bar{t}^d$ denotes the average deadline of the jobs for processing them in a respective CSP. The total computational speed of a CSP needs to be scaled dynamically by creating new VMs so as to complete the jobs within their deadline, whenever the system gets overloaded. The required processing speed of a scalable VM is represented as VM_{req} and it is computed as given in Eq. (10)

After finding A_{ps}, the additional number of VMs required for processing the jobs in J_{f-q} is represented as η and it is computed as given in Eq. (12)

$$\eta = \begin{cases} \left\lceil A_{ps}/VM_{req} \right\rceil; & if \; VM_{req} \le A_{ps} \\ 1; & Otherwise \end{cases} \tag{12}$$

'η' number of VMs are created with the specified processing speed VM_{req}. The processing speed of the scalable VMs also depend on the CSP policy. The scaled VM information is updated in VM_a in the VMM. Finally VMM stores all the information relevant to the VMs. After scaling VMs and prioritizing the jobs in J_{f-q}, the jobs in J_{f-q} are appended with the jobs in J_{s-q}.

4.3.1 One-tier VM architecture

The ARP is similar to the backfilling algorithm and it considers that the state of the job can be saved and restored; as a result the ARP is able to resume its computation in later times with the aid of different VMs. The ARP incurs job preemption and restoring cost in the SHARP approach. The prioritized jobs are stored in the queue J_{s-q}. After prioritizing, the SHARP approach provides customized resources to process the jobs in J_{s-q} when there is enough number of VMs in VM_a. Sometimes, the number of VMs in the VM_a may be inadequate and incapable to process the job at the head of J_{s-q}. Hence, the ARP with the support of backfilling algorithm schedules the subsequent jobs in J_{s-q} that may have the ability to complete within their deadline with the aid of VMs in VM_a by utilizing the idle computing power of the VMs. The job at the head of the J_{s-q} is scheduled for processing whenever the total number of VMs which includes VMs processing low priority jobs, idle VMs and capable VMs in VM_a is greater than or equal to $r^{\min}$ of the job.

Mapping of jobs in a one-tier VM algorithm

```
Input: The jobs in $J_{s-q}$ and $VM_a$
Output: Map the job with the VM
Begin
  Get first 'j' from $J_{s-q}$
  While $j \neq null$ do
    If $j\ satisfy\ (5)\ or\ (7)$ then
      If $\left(\left(\sum_{j\,in\,J_{s-q}} j(r^{\min})\right) \leq |(j_i(S_{\max}) \leq VM_k(ps))|\right)$ then
        If $j(r^{\max}) \leq |(j(S_{\min}) \leq VM_k(ps))|$ then
          Remove 'j' from $J_{s-q}$ and                                 }
          dispatch 'j' to $r^{\max}$ idle and eligible VMs              } \\ Dynamic Provisioning
          Update $VM_a$
        Else
          Remove 'j' from $J_{s-q}$ and dispatch to $r^{\min}$ idle and eligible VMs
          Update $VM_a$
        If $j$ is not at the head of $J_{s-q}$ then
          Insert $j$ into $Q_{backfilling}$
      Else
        $VM_{backfilling}$ denotes running the low priority jobs.
        If $j$ satisfies (5) & (7) by considering VMs process 'j' in $Q_{backfilling}$ then
          Suspend 'j' in $Q_{backfilling}$ and move the job back to $J_{s-q}$ one after other
          according to their ascending priority order of jobs until $j\ satisfy\ (5)\ or\ (7)$
          Remove 'j' from $J_{s-q}$ and dispatch to $r^{\min}$ idle and eligible VMs
          Update $VM_a$
    $j \leftarrow$ get the next job from $J_{s-q}$
end
```

The mapping algorithm optimally maps the jobs with the VMs to support dynamic provisioning of resources based on the system load and availability of the VMs thus minimizing the SLA violations.

4.3.2 Two-tier VM architecture

The jobs may not utilize the entire processing speed of the VMs during their execution. Hence, the VMs are deployed in a two-tier VM architecture so as to improve resource utilization and optimally utilize the processing speed of the VMs. Thus, the computing capacity of a VM is partitioned into foreground and background VM so as to utilize the idle computing power of a VM. The VM processing speed is shared dynamically between foreground and background VM. Initially, the processing speed of the VMs are allocated to foreground VMs and remaining idle computing power are provided to the background VMs. The utilization manager maintains all the information's about the utilization of the foreground VM and background VM. Using these informations, VM switcher assigns jobs to them. In the two-tier VM architecture, the foreground VMs and background VMs have high and low priority respectively. To satisfy the priority of foreground and background VM, job having higher priority runs in the foreground VM and utilize the required processing speed of the VM. Similarly, a job having lower priority runs in the background VM to utilize the remaining processing speed of the VM. So, the VMs deployed in a two-tier VM architecture have the ability to handle two jobs concurrently.

After prioritization, the SHARP approach initially processes the jobs as per the job order in J_{s-q} whenever there are sufficient number of foreground VMs available in VMM. After assigning the jobs to foreground VMs, the expected completion time of the jobs are computed using the attributes of jobs and VMs. The expected completion time of a job running in a particular VM is represented as E^T and is computed as given in Eq. (13).

$$E_j^T = \frac{j\,(S_{\max}) \vee j\,(S_{\min})}{VM\,(ps)}; \quad j\ running\ in\ F_{VM} \tag{13}$$

The ARP chooses $r^{\min}$ or $r^{\max}$ required resources of the running jobs depending on the system load to perform their execution. Sometimes, the number of foreground VMs in VM_a become inadequate and incapable to process the job at the heads of J_{s-q}. At that moment, the ARP with the aid of backfilling algorithm schedules and allocates the subsequent jobs that can complete within their deadline, to the foreground VM. Also, the ARP checks the possibility of assigning the jobs to the background VM by comparing the deadline of the incoming job with the E^T of job in J_{s-q} and E^T of the running jobs of each virtual machine to decide whether the job can be allocated to the same VM. Among the VMs in VMM, those which are eligible to process and complete the jobs within their deadline, are selected for processing the jobs and are represented as VM_{te}. VM_{te} is calculated as given in Eq. (14).

$$VM_{te} = \begin{cases} 1; & if\ \left(E^T_{j(running)} + E^T_j\right) < j\left(t^d\right) \\ 0; & otherwise \end{cases} \quad \forall VM\ in\ VMM \tag{14}$$

The eligible VMs are stored in VM_{temp}. Then, a job in J_{s-q} may be assigned to the background VM only if the job satisfies the constraints defined in Eq. (15). The mapping of a job with the background VM is represented as M^j_{VM} and is given in Eq. (15).

$$M^j_{VM} = \begin{cases} map\ j\ with\ VMs\ in\ VM_{temp}; & if\ \left(j\left(r^{\min}\right) \vee j\left(r^{\max}\right)\right) < \left|VM_{temp}\right| \\ Not\ mapped; & Otherwise \end{cases} \tag{15}$$

While mapping, the resource requirements of the job may vary depending on the system load. The jobs only choose the appropriate number of VMs in the VM_{temp}. In two-tier VM architecture, the delay of running jobs in foreground VMs do not affect the subsequent job processing because the remaining idle processing speed of the VMs are utilized by background VMs for subsequent job execution. Once a job running in the foreground VM completes it's execution, the VM switcher swaps the job running in the background VM to foreground VM to satisfy the priority of foreground and background VM. Further, the job at the head of J_{s-q} is scheduled for processing whenever the total number of VMs that includes VMs that are processing low priority jobs, idle VMs and capable VMs in VM_a is greater than or equal to $r^{\min}$ of a job at the head of the J_{s-q}.

Mapping of jobs in a one-tier VM algorithm

```
Input: The jobs in $J_{s-q}$ and $VM_a$
Output: Map job with VM
Begin
  Get first '$j$' from $J_{s-q}$
  While $j \neq null$ do
      If '$j$' at the head of $J_{s-q}$ then
          Run one tier VM algorithm by considering VMs as foreground VMs
      Else
          Compute $VM_{temp}$ using equation (14)
          Map '$j$' to background VM only after it satisfies equation (15)
      If '$j$' running in foreground VM completes then
          $VM_{backfilling}$ denotes running the low priority jobs.
          Suspend '$j$' in $Q_{backfilling}$ and move the job back to $J_{s-q}$ one after other according to
          their ascending priority order of jobs until '$j$' running in background VM
          Remove '$j$' from background VM and dispatch to $r^{\min}$ of eligible foreground VMs
          Update $VM_a$
  $j \leftarrow$ get the next job from $J_{s-q}$
End
```

The algorithm is initiated when a new job arrives or when the running jobs come to a completion.

5 Experimental setup and performance analysis

Cloud service providers provide service to naïve users based on the storage and computation request. Practically, it is difficult to obtain cloud infrastructure to perform the experiments. For the rationale of research, many open source tools are available to develop and test the experiments. Among these tools, CloudSim is the most popular tool for conducting the simulation experiments [41]. It also provides a generalized and extensible framework for modeling and simulating the proposed algorithm and the efficiency can be compared with other existing algorithms. It simulates the behavior of an IaaS provider by receiving the requests from the naïve users and process them using the VMs.

5.1 Experimental setup

CloudSim toolkit has been selected to simulate the proposed work and verify it's effectiveness. The personal computer configuration is as follows 3.4 GHz, 4 GB RAM, 750 GB hard disk and 64 bit OS, which provides a platform for deploying the CloudSim [42]. The workload traces of grid workload archive (GWA) contains informations about the non-interactive batch jobs [40,43]. Since this work focuses on analyzing and studying information about the cloud users with non-interactive independent batch jobs, GWA meets the common objectives of cloud by providing workload traces that reflect the characteristics of the real application running in one VM. Using GWA workload traces, the job parameters like submission time, requested number of VMs, the actual runtime of applications and VM processor information are not only recognized but also the length of a job (Million Instructions) is computed. Though the workload traces do not have information about the deadline of the jobs, they are randomly generated and an XML file is generated with the help of GWA workload traces along with deadline of the jobs. The proposed SHARP approach has developed a new strategy by integrating job prioritization, scheduling algorithms and resource provisioning techniques. The computing capacity of the data center is considered as 15,000 MIPS (million instructions per second). In a real environment, the number of jobs submitted by the clients vary with respect to time. Hence in the proposed work, the jobs are randomly generated and processed in the cloud environment. Ten runs are

Table 1 Simulation parameters

Parameter	Range
Job size (MI)	5000–25000
Number of jobs	40–800
Maximum number of VMs request in homogeneous environment	60
Maximum number of VMs request in heterogeneous environment	75
Computing power on data center (MIPS)	15,000
Computing power of VM in homogeneous environment (MIPS)	250
Computing power of VM in heterogeneous environment (MIPS)	200–1500
Number of VM	15–75
Deadline (ms)	10–100

carried out with the generated dataset and the number of jobs and VMs are considered as like in the previous iteration. The standard deviation between the previous run and next run is varied approximately between + or − 0.2.

The number of jobs, number of VMs and their corresponding simulation parameters are shown in Table 1. During scaling of VMs, the range of the simulation parameters may vary with respect to system load.

5.2 Performance analysis

Since the main objective is to meet the SLA requirements of the jobs, first the proposed SHARP approach is compared with the existing techniques for their effectiveness in minimizing SLA violations, minimizing the waiting time, minimizing number of VMs and maximizing the data center utilization. The utilization of a data center is indicated by the number of VMs used for a given workload.

5.2.1 Number of VMs

Figure 2 shows the number of VMs required for processing the jobs. This section shows how the number of VMs utilized vary with respect to workload to meet the SLA requirements of the applications and complete them successfully. It can be noticed that the number of VMs utilized by SHARP approach remains approximately stable with respect to workloads. It is absorbed from the experiment that the average number of VMs utilized in the proposed SHARP approach is approximately 30% less than the average of existing algorithms. The reason for such a wide variation is that SHARP approach tries to run batch jobs by using unutilized VMs with the help of adaptive resource provisioning when sparse number of jobs are submitted. When more number of batch jobs are submitted, the APR maintains stability by allocating the minimum number of resources depending on the system load. It is observed that during overloading, the servers used by the existing approach are scaled up thus increasing the number of VMs up to 50. This is due to static scalability of server. In SHARP approach, if any running job delays its execution, the remaining idle computing power is used to run the subsequent jobs in a background VM.

Figure 2a, b represents the number of VMs required for processing the jobs in a homogeneous environment and heterogeneous environment respectively. The maximum number of VMs in a homogeneous environment is 60 and have equal processing speed. But in a heterogeneous environment, the maximum number of VMs created varies between 15 and 75. The jobs are mapped with appropriate VMs in a heterogeneous environment. The SHARP approach minimizes the number of VMs utilized in two-tier compared to one-tier VM architecture by utilizing the idle computing power of the VMs.

5.2.2 Waiting time of the jobs

To make a precise study and analysis of the proposed approach, the jobs are randomly submitted in batches by partitioning the jobs in workload traces that vary with time as in a cloud environment. The current scheduling algorithms concentrate only on prioritizing the jobs submitted by the naïve users, but are not aware of mapping them with the VMs in the cloud system so as to complete them successfully within their deadline. To the exceptional of our talents, the present resource provisioning algorithms do not support dynamic resource provisioning and so the VMs within the VMM may be insufficient or surplus depending upon the system load. Moreover, the present schedulers can only optimally schedule uniprocessor requests. But, many problematic applications demand multiprocessor to perform their computation. So, SHARP approach has been proposed to execute both uniprocessor as well as multiprocessor requests of the naïve users.

Figure 3 represents the waiting time of the jobs. Figure 3a, b represents the waiting time of the jobs in homogeneous environment and heterogeneous environment respectively. The SHARP approach integrates preprocessing to eliminate the unfeasible jobs so as to minimize the waiting time of the jobs, does job prioritization with the aid of job and resource

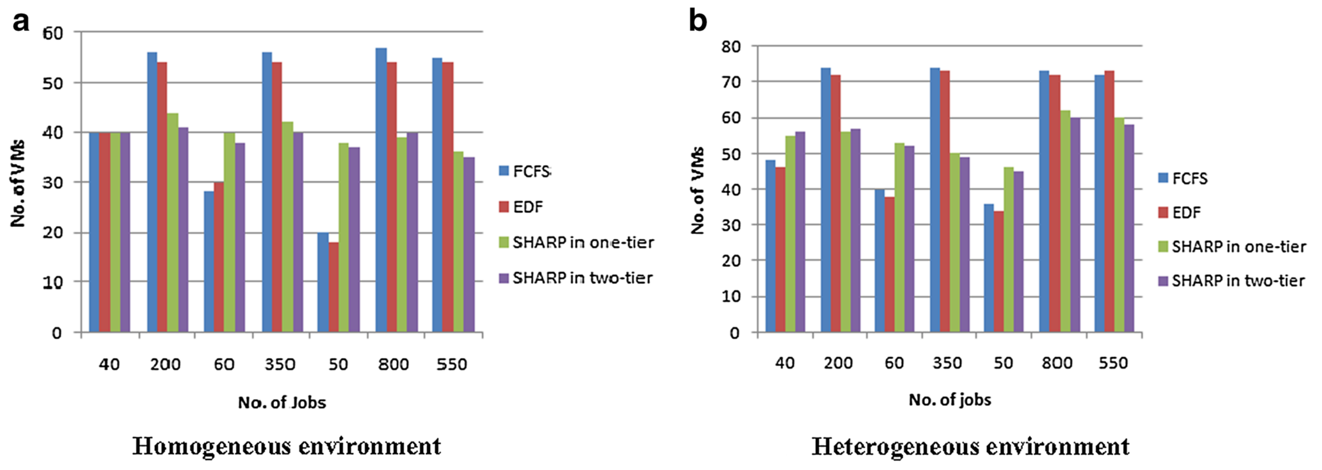


Fig. 2 Comparison of VMs required in various approaches

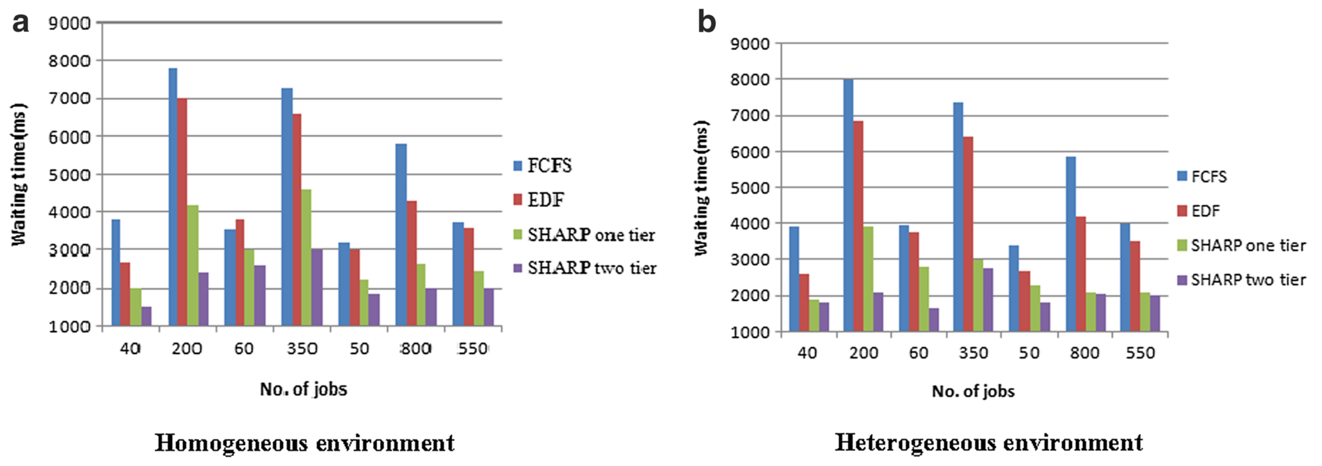


Fig. 3 Waiting time of the jobs. **a** Homogeneous environment. **b** Heterogeneous environment

attributes, and provides resources adaptively. In adaptive resource provisioning, the new VMs are created depending upon the system load. SHARP approach integrates backfilling algorithm to schedule the subsequent jobs when the VMs required to process ‘j’ at the head of J_{s-q} is greater than the capable VMs in VMM. In SHARP approach, the waiting time of a job is further reduced by deploying the VMs in a two-tier architecture which avoids process delay caused by running jobs on subsequent jobs, by running them in a background VM. During overloading, the SHARP approach scales up the required number of VMs and thus minimizes the waiting time of the jobs.

5.2.3 Resource utilization

Since the main objective of the scheduler is to maximize the revenue of the service provider by optimally utilizing them, the proposed approach concentrates on data center utilization. The data center utilization is indicated by the number of VMs which are used for processing the given workload. Figure 4 represents the resource utilization. It can be observed that the number of VMs utilized vary with respect to workload and their deadline. It is also noticed that the proposed approach utilized the VMs efficiently by partitioning the computing capacity of VMs into foreground and background VMs. The batch jobs always run in the foreground VM. If the jobs cannot use the full processing speed of the VM, the subsequent jobs may run in the background VM by utilizing the remaining idle computing power of the VM. Hence, the proposed approach uses less number of VMs compared to other techniques thus improving the utilization of data center by 10–15 %.

Figure 4a, b represents the resource utilization in a homogeneous and heterogeneous environment respectively. The SHARP approach integrates backfilling algorithm to boost the resource utilization by running the succeeding jobs when the ‘j’ at the head of J_{s-q} requires more VMs than the capable VMs in VMM. During a sparse number of requests, the SHARP approach picks the highest possible number of VMs to stabilize the system utilization which boosts the resource

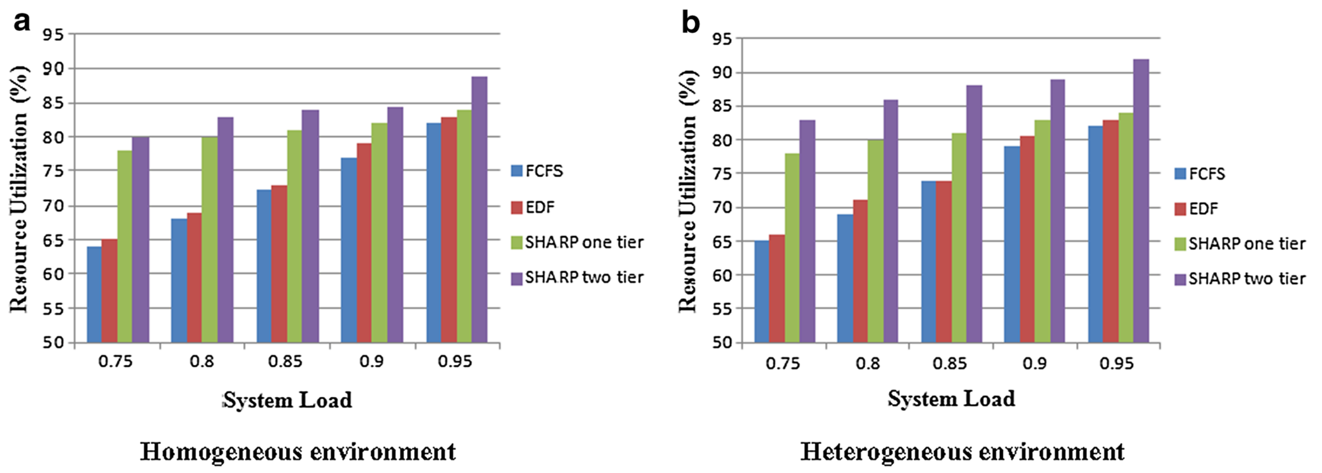


Fig. 4 Resource utilization. **a** Homogeneous environment. **b** Heterogeneous environment

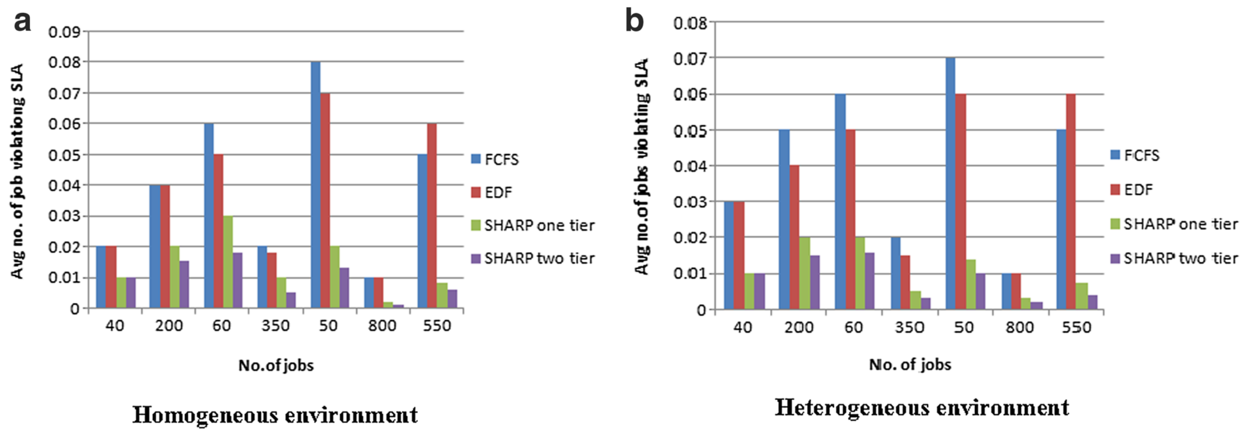


Fig. 5 SLA violations. **a** Homogeneous environment. **b** Heterogeneous environment

utilization and improves user satisfaction. On the other hand, the SHARP approach chooses the jobs with minimum number of VMs request to avoid over provisioning of VMs. Thereby, the proposed SHARP approach totally increases the utilization of the resources by 25% to 30 %.

5.2.4 SLA violations

This section evaluates how the deadline of the batch jobs affects the performance of service providers. Even though the number of VMs is unlimited in real cloud infrastructure, this experiment initially limits the number of VMs to 75 and the number of batch jobs vary between 50 and 800. Fixing the number of VMs and batch jobs range allowed to observe the effects of the deadline with the system performance. Figure 5 represents the SLA violations. As the size of the workload increases and users' urgency level increases, SHARP approach dramatically minimizes the number of SLA violations from 10 to 55% by dynamically scaling up the VMs depending on the system load.

Figure 5a, b represents the SLA violations of job in homogeneous and heterogeneous environment respectively. The reason for such a large difference is that the proposed scheduler could initiate new VMs when there is more number of batch job submissions. The number of jobs submitted by the naïve users vary with respect to time. Though the existing algorithms do not allocate the resources based on the system load, the proposed SHARP approach supports dynamic provisioning of resources depending upon the system load and job requirements. The proposed SHARP approach minimizes the SLA violations by scaling up the number of VMs depending on the system load.

6 Conclusion and future work

The number of jobs submitted by the naïve users vary with respect to time. The ultimate objectives of the work were to fulfill the user requirements, to stabilize the number of VMs, to mitigate SLA violations of jobs, and to boost resource utilization. The proposed research work presents ScHeduling of jobs and Adaptive Resource Provisioning (SHARP) for optimal scheduling and processing of deadline based independent jobs in a cloud environment. The proposed SHARP approach examines the difficulties of current scheduling algorithms and bilaterally focuses on job scheduling and resource provisioning. In SHARP approach, MCDA preprocesses the jobs using the different attributes of jobs and resources to mitigate the number of jobs violating their SLA requirements. Moreover, MAJS not only prioritizes the uniprocessor jobs but also prioritizes multiprocessor jobs dynamically. After prioritizing, the jobs are optimally processed with the help of ARP which provides resources depending upon the utilization of the system and requirements of jobs to stabilize the system load. Further, the ARP approach scales up the number of VMs when the existing VMs in VMM are not able to process and complete the jobs within their deadline. Moreover, ARP integrates backfilling algorithm to boost the resource utilization. Moreover, the VMs are deployed in a two-tier VM architecture to utilize the idle processing speed of the VMs. The VM switcher allocates the jobs among foreground and background VM based on the priority of the jobs. The SHARP approach supports for both homogeneous as well as heterogeneous environment.

The SHARP approach outperforms other existing algorithms by mitigating the number of jobs violating their deadline to improve user satisfaction and by boosting resource utilization with the aid of elastic resource provisioning that stabilizes system load. Moreover, the SHARP approach minimizes the number of active VMs to process the incoming jobs and reduces the waiting time of the jobs. In future, this work can be extended to develop an efficient scheduler by taking into consideration data transfer time for processing both dependent and independent jobs. Further, the proposed SHARP approach can be integrated with e-learning systems to optimally schedule and manage the e-learning workloads in a cloud environment.

References

1. Jain, R., Paul, S.: Network virtualization and software defined networking for cloud computing: a survey. IEEE Commun. Mag. **51**(11), 24–31 (2013)
2. Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z.: The characteristics of cloud computing. In: 2010 39th International Conference on Parallel Processing Workshops, pp 275–279 (2010)
3. Payberah, A.H., Kavalionak, H., Kumaresan, V., Montresor, A., Haridi, S.: Clive: cloud-assisted p2p live streaming. In: 2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P), IEEE, pp 79–90 (2012)
4. Li, C., Raghunathan, A., Jha, N.K.: A trusted virtual machine in an untrusted management environment. IEEE Trans. Serv. Comput. **5**(4), 472–483 (2012)
5. Garg, S.K., Yeo, C.S., Anandasivam, A., Buyya, R.: Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. J. Parallel Distrib. Comput. **71**(6), 732–749 (2011)
6. Zhou, A., Sun, Q., Sun, L., Li, J., Yang, F.: Maximizing the profits of cloud service providers via dynamic virtual resource renting approach. EURASIP J. Wirel. Commun. Netw. **2015**(1), 1–12 (2015)
7. Maguluri, S.T., Srikant, R., Ying, L.: Stochastic models of load balancing and scheduling in cloud computing clusters. In: INFOCOM, 2012 Proceedings IEEE, IEEE, pp 702–710 (2012)
8. Zhao, C., Zhang, S., Liu, Q., Xie, J., Hu, J.: Independent tasks scheduling based on genetic algorithm in cloud computing. In: 2009 5th International Conference on Wireless Communications, Networking and Mobile Computing, pp 1–4 (2009)
9. Masdari, M., ValiKardan, S., Shahi, Z., Azar, S.I.: Towards workflow scheduling in cloud computing: a comprehensive analysis. J. Netw. Comput. Appl. **66**, 64–82 (2016)
10. Rajavel, R., Thangarathanam, M.: Adaptive probabilistic behavioural learning system for the effective behavioural decision in cloud trading negotiation market. Future Gener. Comput. Syst. **58**, 29–41 (2016)
11. Yao, G., Ding, Y., Jin, Y., Hao, K.: Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system. Soft Comput., 1–14 (2016). doi:10.1007/s00500-016-2063-8
12. Komarasamy, D., Muthuswamy, V.: A novel approach for dynamic load balancing with effective bin packing and vm reconfiguration in cloud. Indian J. Sci. Technol. **9**(11), 1–6 (2016)
13. Zhu, J., Li, X.: Scheduling for multi-stage applications with scalable virtual resources in cloud computing. Int. J. Mach. Learn. Cybern., 1–9 (2016). doi:10.1007/s13042-016-0533-z
14. Wu, F., Wu, Q., Tan, Y., Wang, W.: Unified multi-constraint and multi-objective workflow scheduling for cloud system. In: Algorithms and Architectures for Parallel Processing, pp 635–650. Springer, Berlin (2015)
15. Ye, H.: Research on emergency resource scheduling in smart city based on HPSO algorithm. Int. J. Smart Home **5**, 6 (2015)
16. Sheikhalishahi, M., Wallace, R.M., Grandinetti, L., Vazquez-Poletti, J.L., Guerriero, F.: A multi-dimensional job scheduling. Future Gener. Comput. Syst. **54**, 123–131 (2016)
17. Nathani, A., Chaudhary, S., Somani, G.: Policy based resource allocation in IaaS cloud. Future Gener. Comput. Syst. **28**(1), 94–103 (2012)
18. Huang, Y., Bessis, N., Norrington, P., Kuonen, P., Hirsbrunner, B.: Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm. Future Gener. Comput. Syst. **29**(1), 402–415 (2013)
19. Ahmad, A., Arshad, R., Mahmud, S.A., Khan, G.M., Al-Raweshidy, H.S.: Earliest-deadline-based scheduling to reduce urban traffic congestion. IEEE Trans. Intell. Transp. Syst. **15**(4), 1510–1526 (2014)
20. Van den Bossche, R., Vanmechelen, K., Broeckhove, J.: Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. Future Gener. Comput. Syst. **29**(4), 973–985 (2013)
21. Lee, J., Shin, K.G.: Preempt a job or not in EDF scheduling of uniprocessor systems. IEEE Trans. Comput. **63**(5), 1197–1206 (2014)
22. Li, J., Luo, Z., Ferry, D., Agrawal, K., Gill, C., Lu, C.: Global edf scheduling for parallel real-time tasks. Real Time Syst. **51**(4), 395–439 (2015)
23. Tang, Z., Zhou, J., Li, K., Li, R.: A mapreduce task scheduling algorithm for deadline constraints. Clust. Comput. **16**(4), 651–662 (2013)
24. Abrishami, S., Naghibzadeh, M., Epema, D.: Cost-driven scheduling of grid workflows using partial critical paths. In: 2010 11th IEEE/ACM International Conference on Grid Computing, pp 81–88 (2010)

25. Abrishami, S., Naghibzadeh, M., Epema, D.H.J.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013)
26. Calheiros, R.N., Buyya, R.: Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans. Parallel Distrib. Syst. **25**(7), 1787–1796 (2014)
27. Komarasamy, D., Muthuswamy, V.: Deadline constrained adaptive multilevel scheduling system in cloud environment. TIIS **9**(4), 1302–1320 (2015)
28. Liu, X., Wang, C., Zhou, B.B., Chen, J., Yang, T., Zomaya, A.Y.: Priority-based consolidation of parallel workloads in the cloud. IEEE Trans. Parallel Distrib. Syst. **24**(9), 1874–1883 (2013)
29. Liu, Y., Zhang, C., Li, B., Niu, J.: Dems: a hybrid scheme of task scheduling and load balancing in computing clusters. J. Netw. Comput. Appl. **83**, 213–220 (2015)
30. Madni, S.H.H., Latiff, M.S.A., Coulibaly, Y.: Resource scheduling for infrastructure as a service (iaas) in cloud computing: challenges and opportunities. J. Netw. Comput. Appl. **68**, 173–200 (2016)
31. Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. **25**(3), 682–694 (2014)
32. Lee, Y.-H., Leu, S., Chang, R.-S.: Improving job scheduling algorithms in a grid environment. Future Gener. Comput. Syst. **27**(8), 991–998 (2011)
33. Zhang, J., Huang, H., Wang, X.: Resource provision algorithms in cloud computing: a survey. J. Netw. Comput. Appl. **64**, 23–42 (2016)
34. Somasundaram, T.S., Govindarajan, K.: Cloudrb: a framework for scheduling and managing high-performance computing (HPC) applications in science cloud. Future Gener. Comput. Syst. **34**, 47–65 (2014)
35. Krishnamoorthy, N., Asokan, R.: Hybrid adaptive job and resource scoring meta-scheduling system for grid computing. J. Theor. Appl. Inf. Technol. **54**(3), 444–452 (2013)
36. Morariu, O., Morariu, C., Borangiu, T.: A genetic algorithm for workload scheduling in cloud based e-learning, In: Proceedings of the 2nd International Workshop on Cloud Computing Platforms, p. 5. ACM, New York (2012)
37. Al-Ayyoub, M., Jararweh, Y., Daraghmeh, M., Althebyan, Q.: Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure. Cluster Comput. **18**(2), 919–932 (2015)
38. Komarasamy, D., Muthuswamy, V.: Associate scheduling of mixed jobs in cloud computing. In: Proceedings of the 3rd International symposium on Big data and Cloud Computing Challenges (ISBCC 16), pp. 133–142. Springer, Cham (2016)
39. Roy, S., Banerjee, S., Chowdhury, K., Biswas, U.: Development and analysis of a three phase cloudlet allocation algorithm. J. King Saud Univ. Comput. Inf. Sci. (2016). doi:10.1016/j.jksuci.2016.01.003
40. Garg, S.K., Toosi, A.N., Gopalaiyengar, S.K., Buyya, R.: Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter. J. Netw. Comput. Appl. **45**, 108–120 (2014)
41. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
42. Kong, W., Lei, Y., Ma, J.: Virtual machine resource scheduling algorithm for cloud computing based on auction mechanism. Optik Int. J. Light Electron Opt. **127**(12), 5099–5104 (2016)
43. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H.: The grid workloads archive. Future Gener. Comput. Syst. **24**(7), 672–686 (2008)

Dinesh Komarasamy is doing his research scholar in the Department of Information Science and Technology, College of Engineering, Anna University. He has obtained his M.E graduate in the specification of Computer Science and Engineering and B.E graduate in the specification of Computer Science and Engineering during the year 2012 and 2010 respectively. His research area include Cloud Computing and Optimization Techniques. He has presented 4 research publications in conferences and published 7 papers in the reputed journals. Currently he is an Assistant Professor in the Department of Computer Science and Engineering, Kongu Engineering College, Anna University, India.

Vijayalakshmi Muthuswamy is currently working as Assistant Professor (Sr. Gr) in Anna University, Chennai. She received her M.E and Ph. D from Anna University. She has published 50 articles in journals and conferences. Her area of interest includes Computer Networks, Mobile Cloud, Cloud Computing and Security.