Deadline Constrained Adaptive Multilevel Scheduling System in Cloud Environment

Dinesh Komarasamy[1], Vijayalakshmi Muthuswamy[2]
[1] *Department of Information Science and Technology, CEG Campus, Anna University*
Chennai- 600025, India
[e-mail: dinesh.nova@gmail.com]
[2] Department of Information Science and Technology, CEG Campus, Anna University
Chennai- 6000025, India
[e-mail: vijim@annauniv.edu]
*Corresponding author: Dinesh Komarasamy

Received November 15, 2014; revised February 22, 2015; accepted March 7, 2015; published April 30, 2015

Abstract

In cloud, everything can be provided as a service wherein a large number of users submit their jobs and wait for their services. Thus, scheduling plays major role for providing the resources efficiently to the submitted jobs. The brainwave of the proposed work is to improve user satisfaction, to balance the load efficiently and to bolster the resource utilization. Hence, this paper proposes an Adaptive Multilevel Scheduling System (AMSS) which will process the jobs in a multileveled fashion. The first level contains Preprocessing Jobs with Multi-Criteria (PJMC) which will preprocess the jobs to elevate the user satisfaction and to mitigate the jobs violation. In the second level, a Deadline Based Dynamic Priority Scheduler (DBDPS) is proposed which will dynamically prioritize the jobs for evading starvation. At the third level, Contest Mapping Jobs with Virtual Machine (CMJVM) is proposed that will map the job to suitable Virtual Machine (VM). In the last level, VM Scheduler is introduced in the two-tier VM architecture that will efficiently schedule the jobs and increase the resource utilization. These contributions will mitigate job violations, avoid starvation, increase throughput and maximize resource utilization. Experimental results show that the performance of AMSS is better than other algorithms.

Keywords: Cloud Computing, Job Scheduling, Priority Scheduler, Load Balancing, Resource Utilization.

http://dx.doi.org/10.3837/tiis.2015.04.002 **ISSN : 1976-7277**

1. Introduction

Today, Internet is more than just a communication medium. It has been moved into the era of e-commerce and e-governance. The number of users accessing Internet is increasing exponentially that raises the demand for developing advanced network technologies like cloud computing, big data and mobile cloud computing. Every day, the Internet's infrastructure is dramatically changing and adapting itself to the heterogeneous networks with the help of cloud for providing high end services like Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) as elucidated by M. D. Dikaiakos et al., and K. Dinesh et al. [1, 2]. Internet acts as a bridge between service providers and users for providing service. The general characteristics of cloud computing are to provide on-demand service, broad network access, resource pooling, rapid elasticity and measured service to the user as explained by P. Mell and T. Grance [3]. Cloud computing is usually called as pay per usage model as explained by Md. Sabbir Hasan and Eui-Nam Huh [4]. It reduces user expenditures on hardware, software and maintenance cost. It also provides reliable, highly available and time critical service to the users as explored by Daeyong Jung [5].

In cloud computing, large numbers of user submit their jobs to the cloud broker who transfers the request to the Cloud Service Providers (CSP) as explained by Kwang Mong Sim [6]. CSP provides the services transparently to the users independent of host infrastructure through virtualization. Virtualization is a technique that logically separates the physical resource. Each logical unit of physical resource acts as a VM. The necessity of virtualization is to provide hardware independence, software isolation, reduce energy consumption and security with increased resource utilization as explained by Li. Chunxiao et al. [7]. The jobs, if not properly scheduled in the cloud environment lead to network congestion. Therefore, more numbers of jobs are discarded due to network congestion. A good scheduling algorithm should speed up the job execution to reduce network traffic. Consequently, the user satisfaction and the number of jobs accepted for execution increases that will boost the revenue of the CSP and also reduce the local network traffic.

Various scheduling algorithms [8-24] were developed for scheduling the jobs in the cloud environment. Among these algorithms, lots of heuristic algorithms alter their scheduling policies depending on the type of jobs or nature of resources. Hence, these algorithms may not be suitable for scheduling in the dynamic cloud environment. Many scheduling algorithms have been developed and demonstrated for efficient scheduling of deadline based jobs in a dynamic environment. Though these algorithms only focus on downright the job execution within its deadline constraint for improving user satisfaction, they do not concentrate on resource utilization and load balancing.

To schedule the jobs and to balance the load effectively, this paper proposes an Adaptive Multilevel Scheduling System (AMSS). The AMSS processes the jobs in multileveled manner, which composed of four major components as Preprocessing Jobs with Multi-Criteria (PJMC), Deadline Based Dynamic Priority Scheduler (DBDPS), Contest Mapping Jobs with Virtual Machine (CMJVM) and VM Scheduler (VMS). The jobs are initially preprocessed in PJMC by deeming multiple criteria. The priorities are dynamically assigned to the jobs in the next phase (i.e. DBDPS) after preprocessing. The prioritized jobs are dynamically mapped to the appropriate VM using CMJVM. The VM Scheduler dynamically schedules the jobs between FVM and BVM. The proposed work reduces the average number of job violations and also increases resource utilization.

The rest of the paper is organized as follows. Section 2 discusses about the related work. Section 3 describes about the characteristics of the system model. Section 4 illustrates about the design of an Adaptive Multilevel Scheduling System for scheduling the jobs efficiently. Section 5 explains the simulation and results analysis of the proposed work. Section 6 gives conclusion remarks and future enhancements.

2. Related Work

This section reviews various multifarious scheduling algorithms developed to schedule the jobs depending on the type of job or resource in a cloud environment. The jobs are classified as batch jobs, transactional jobs and interactive jobs based on their characteristics as explained by Y. Zhang, et al. and D. Carrera, et al. [8, 9]. The proposed work spotlights on batch job scheduling and hence the literatures have been restricted to batch job scheduling. The batch job scheduling is categorized into two types as static and dynamic scheduling depending upon the characteristics of scheduling. In static scheduling, the jobs, which are executed in certain resources, are also non-preemptive. In static scheduling, jobs are mostly processed using First Come First Serve (FCFS) model. In FCFS, long running jobs affect small running jobs during their execution as elucidated by A. Silberschatz, et al. [10]. Static scheduling is not much suitable for cloud computing because the jobs are aperiodic. Unlike static scheduling, the jobs are scheduled at the run time in dynamic scheduling that supports migration and preemption.

An advanced reservation technique was developed for executing jobs where the scheduler in this technique does an advanced reservation by considering additional information on jobs such as the starting time, execution time and processing speed requested by the job as explicated by D. Nurmi et al. [11]. The resources are underutilized due to the effect of advanced reservation technique. Therefore, Backfilling algorithm was used to improve resource utilization. Backfilling algorithm is generally called as an optimized technique of FCFS that improves the performance of the system. In backfilling algorithm, the small jobs were run without affecting the waiting job, due to the non-availability of sufficient resources, in the head of the queue as modeled by Y. Zhang, et al. and A. W. Mu'alem, et al. [8, 12].

The jobs are generally classified into two types such as deadline based and non-deadline based jobs depending upon the user input. Deadline based jobs are scheduled using Earliest Deadline First (EDF) algorithm to complete earliest deadline jobs within their deadline as developed by V. Gamini Abhaya, et al. [13]. EDF is a type of priority scheduling. In order to complete the jobs within the deadline, a sub-deadline is assigned and distributed to all nodes using a Partial Critical Path algorithm (PCP). PCP contains two phases like deadline distribution and planning phase for completing the jobs within the deadline as explained by S. Abrishami, et al.[14]. The jobs are prioritized based on not only deadline, but also the arrival time, waiting time and so on. The jobs are dynamically prioritized and mapped to the VMs with limited support of migration in dynamic scheduling. The job preemption and job migration can fritter away execution time and network bandwidth as developed by M. Stillwell, et al. [15].

Various heuristic algorithms have been introduced for job scheduling with different problem constraints. Heuristic algorithms were also exploited for finding out the degree of matching between the jobs and resources as elucidated by R. Baraglia, et al. [16]. For mapping jobs to the resources, the Berger model stated a set of definition like task justice or injustice, system justice and integrated justice function. Berger model was developed based on commodity economic model and market mechanism as developed by Baomin Xu, et al. [17].

Here, the jobs are scheduled and mapped with the VMs present in the resource pool. The VMs may exist either in a homogeneous or heterogeneous environment as explained by Ehsan Ullah Munir, et al. [18].

A Hierarchical Load balanced algorithm (HLBA) was developed for scheduling the jobs in a hierarchical framework. It schedules the incoming jobs to the cluster having the fastest idle computing power without worrying about the average load of the system. In HLBA, the incoming jobs are scheduled based on their weighted value. The weighted value has been determined using network utilization, memory utilization and idle CPU processing power as explained by Yun-Han Lee, et al. [19]. The resource utilization was further increased using Resource Attribute Selection (RAS) algorithm. In RAS algorithm, the jobs are allocated to the resource computing capacity, storage space and network utilization of the node [20].

The arrival rate and service rate are not easy to predict and maintain for large scale cloud, various scheduling algorithms were developed to schedule the jobs. But, a few algorithms like blind online scheduling algorithm (BOSA) were proposed to schedule the request without knowing the arrival rate and service rate. These requests were forwarded to the server having the large free time slot service to minimize the waiting time and load balancing using FCFS model as explicated by Liang Zhou, et al. [21, 22].

However, the processing speed of VM cannot be fully utilized by a job during its execution due to the communication delay. Therefore, the two-tier VM architecture was proposed to utilize the idle processing speed of the VM and hence improving the resource utilization. The two-tier VM architecture contains Foreground VM (FVM) and Background VM (BVM) that are pinned to single processing element as illuminated by Xiaocheng Liu, et al. [23]. The incoming jobs are allocated using an EASY backfilling algorithm. The incoming jobs may request either a single processor VM or a multiple processor VM. The incoming job request for a single processor so that the utilization rate of the VM varies between the range of 80% to 92% during the execution of jobs as explained by Xiaocheng Liu, et al. [24]. Otherwise, the utilization of the VM is decreased due to their communication delay and synchronization time interval. Here, the utilization of resource varies between 19.8% and 76.6% [11, 24]. From the review of existing job scheduling, we observed that communication and computation overhead of the resource reduce the performance of the system, user satisfaction and resource utilization. The proposed Adaptive Multilevel Scheduling System (AMSS) effectively schedules the jobs to increase the user satisfaction, resource utilization and mitigates the jobs violating their deadline.

3. Characteristics of System model

3.1 Characteristic of Jobs

The incoming jobs are assumed as batch jobs in this work. The properties of batch jobs are aperiodic (i.e. the arrival time of the job is not known in advance) and independent of each other (i.e. the input of one job does not depend on the output of other jobs) as modeled by Chenhong Zhao, et al. [25]. The incoming jobs are estimated as non-preemptive (i.e. even if a high priority job arrives, the job in execution is not preempted). The jobs (j_i) are defined as $J = \{j_1, j_2, ..., j_N\}$ and their relation is described in equation (1).

$$j_1 \perp j_2 \perp ... \perp j_N ; \ \forall_j \in J \qquad \text{where N represents the number of jobs} \tag{1}$$

Let, $\perp$ represents the independent relation between jobs. It is assumed that the user must specify the length (l) and the corresponding deadline (d) of the job during submission $j_i = \{ls_i, d_i\}; i \in (1, N)$. The length of the job or the size of the job is expressed as the number of instructions required for processing the job. It is generally defined as number of Million Instructions (MI) required for processing the submitted job [26]. The jobs may request either computational resources or storage resources. In this work, it is assumed that the job request for only computational resources for their execution.

3.2 Characteristics of Resources

The resources (i.e. VMs) are independent of each other. VMs may exist either in homogeneous or in heterogeneous environments. The processing speed of the VMs in a homogeneous environment is defined as $S_e = \{S_1, S_2, ..., S_\alpha\}$. Here, all VMs have equal processing speed such that $S_{i-j} = S_i = S_{i+j}$. Similarly, the processing speed of the VMs in a heterogeneous environment is defined as $S_u = \{P_1, P_2, ..., P_\beta\}$ so that all VMs have different processing capacity $P_{i-j} \neq P_i \neq P_{i+j}$. The processing speeds of the VMs are not shared among them as defined below [14].

$$\left.\begin{array}{l}(S_1 = S_2 = ... = S_\alpha) \,\&\, (S_1 \cap S_2 \cap ... \cap S_\alpha = \phi); \text{ hom}\,ogeneous\,environment \\ (P_1 \neq P_2 \neq ... \neq P_\beta) \,\&\, (P_1 \cap P_2 \cap ... \cap P_\beta = \phi); \; heterogeneous\,environment\end{array}\right\} VM \qquad (2)$$

3.3 Problem Statement

In the above scenario, a large number of users submit their jobs in the cloud. Among them, some may request more processing speed than the available processing speed of the VM that may affect the subsequent jobs. These jobs are filtered to reduce the number of jobs violating their deadline and also to increase the performance of the system. Every job contains two attributes namely length and deadline during its submission. The scheduler can effectively schedule and complete the large scale of jobs within their deadline by prioritizing the jobs. The priority scheduler can calculate the priority value based on different parameters like waiting time of the job, length of the job and deadline of the submitted jobs. It does not focus on resource utilization and previous workload. The priority scheduler can efficiently schedule the jobs to the underlying VM so that it can reduce the waiting time (W_t) of the jobs. Thus, it also increases the throughput (T_p) of the system. The relationship between waiting time, throughput and resource utilization (R_u) are described in equation (3).

$$T_p \propto R_u \xrightarrow{Transitive\ \text{Pr}\,operty} T_p \propto \frac{1}{W_t} \qquad (3)$$

The jobs are non-preemptive in nature, so it would execute only once in a particular VM. The processing speed of the VM is generally given as $PS = \{S_e, S_u\}$. Large numbers of VMs $\left(\bar{y} = \left[\left|S_\alpha\right| \vee \left|S_\beta\right|\right]\right)$ are available for processing the jobs. The job is dynamically mapped to a suitable VM based on different parameters like Expected Processing Speed of the submitted job (EPS), previous workload of the VM and the processing speed of the VM. After mapping, the job will be executed in the corresponding VM. During the execution, the job cannot utilize

the full processing speed of the VM due to its communication, synchronization and some other external delay. Therefore, the two-tier VM architecture (i.e. FVM and BVM) has been implemented to utilize the remaining idle processing speed of the VM only if the job does not utilize the full processing speed of the VM. Therefore, a new scheduler is required for scheduling the job between FVM and BVM. Hence, this work proposes an Adaptive Multilevel Scheduling System (AMSS) that will filter, prioritize and map the job to a suitable VM. The proposed work will improve system performance and resource utilization.

4. Design of Adaptive Multilevel Scheduling System

The jobs are submitted from various users with different demand and the jobs are assumed to be demanding only computing resources for their execution. The submitted jobs are congregated to the Cloud User Interface (i.e. Portal). The minimum processing speed required by the job is dynamically estimated depending on the user input. Sometimes, the job may require more processing speed than the available processing speed of the resource. In this work, the jobs are accepted only if it can adapt and complete within deadline using the available processing speed of the VM and eliminate the job that require more processing speed than the available processing speed of the resource. By taking this into account, this paper proposes an Adaptive Multilevel Scheduling System (AMSS) as depicted in **Fig. 1**. The jobs are passed to the AMSS components.

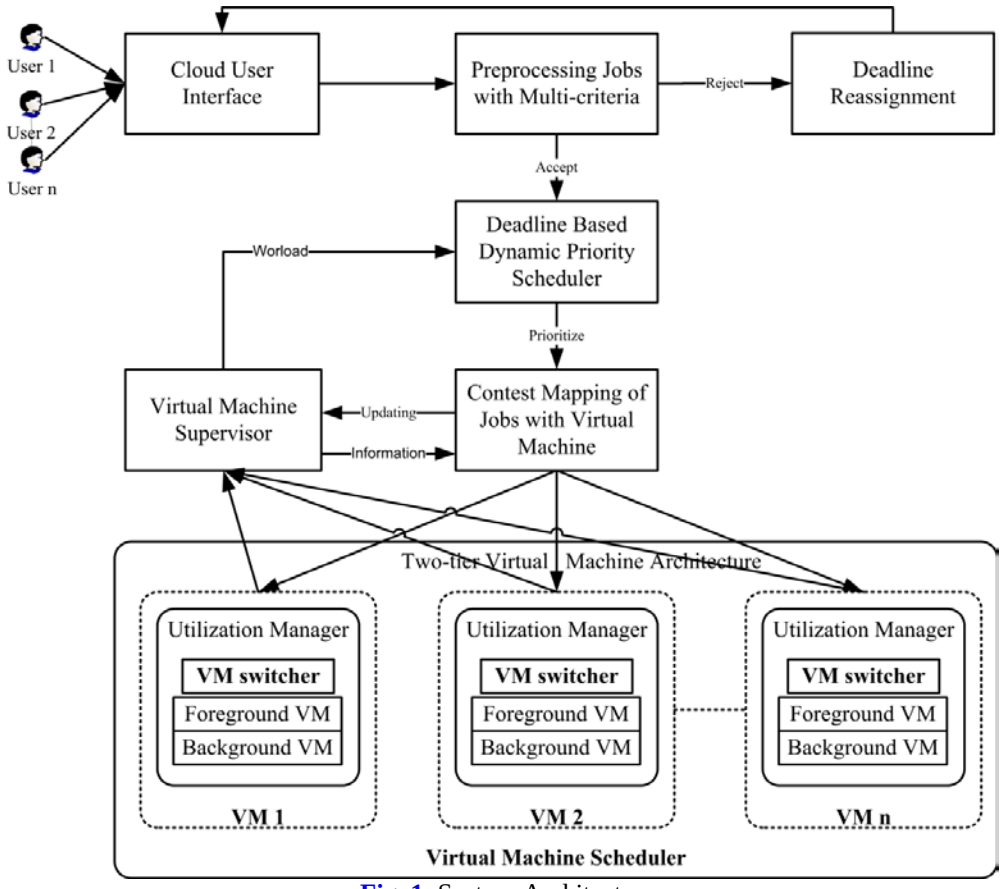


Fig. 1. System Architecture

The AMSS comprises of four components for scheduling the jobs in multilevel manner. In PJMC, preprocessing and filtering are done on the jobs that require more processing speed than the available processing speed of the VM. These jobs are handed over to the deadline reassignment. In the deadline reassignment, the deadline of the job is reassigned only after receiving further requests from the user. The jobs are passed to the second level of AMSS only if the processing speed required by the jobs is within the available processing speed of the VM. In the second level, DBDPS assigns priority dynamically to the jobs. The prioritized jobs are then passed to the next level of AMSS. In the next level, CMJVM dynamically maps the job to suitable resource for completing its execution within deadline. Moreover, CMJVM also balances the system load. After resource allocation, the license is provided to the job for executing in a particular VM for a certain time period. After license assignment, the CMJVM updates the current information of the VM to the VM Supervisor. The VM supervisor maintains the current status and availability of the VM. Finally, the jobs are passed to the VM scheduler. In VM scheduler, the jobs are efficiently scheduled between the FVM and BVM in two-tier VM architecture.

4.1 Preprocessing Jobs with Multi-Criteria (PJMC)

The VM exists either in homogeneous or heterogeneous environment depending on the data center policy $VM = \left(\{S_e\} \vee \{S_u\}\right)$. Since two functions such as either accepted or rejected are carried out, the jobs processed in the PJMC are considered as a Bernoulli distribution (i.e. acceptance is treated as success and rejection is treated as a failure) as each job has two possible outcomes and independent of each other. The minimum processing speed or computation speed required for the submitted job can be represented as S_r and can be calculated as shown in equation (4).

$$S_r^j = \frac{ls_j}{d_j} | \forall_j \in J;\ j \in (1, N) \tag{4}$$

'ls' and 'd' stand for the length and deadline of the job. The processing speed of the VM is expressed in MIPS (Million Instructions Per Second). The maximum processing speed of existing VM is represented as Max_s and is manipulated as depicted in equation (5).

$$Max_s = \begin{cases} S_1 \mid S_1 \in S_e & ;\text{hom}\, ogeneous\, environment \\ \max(P_i \mid \forall_{VM} \in S_u) & ;heterogeneous\, environment \end{cases} \tag{5}$$

The jobs that are accepted in PJMC by comparing S_r and Max_s are represented as J_{f-PJMC} and is expressed in equation (6).

$$J_{f-PJMC} = \begin{pmatrix} 1; if\ S_r \le Max_s \\ 0; otherwise \end{pmatrix} \mid \forall_j \in (1, N) \tag{6}$$

The number of jobs processed in PJMC is considered as a binomial distribution as 'N' trails (i.e. 'N' jobs). Among these trails, '$\overline{N}$' successes (i.e. '$\overline{N}$' jobs) are accepted and stored

in the queue $\left(J_{fq}\right)$. Otherwise, the jobs are rejected and passed to deadline reassignment. The total processing speed of the VM is denoted as S_t and is calculated as shown in equation (7).

$$S_t = \left(\sum_{i=1}^{\alpha} S_i \mid S_i \in S_e\right) \vee \left(\sum_{i=1}^{\beta} P_i \mid P_i \in S_u\right) \tag{7}$$

The jobs are preprocessed based on the Time Required for Processing already accepted jobs (TRP) in a queue and it is computed as given in equation (8).

$$TRP = \frac{\sum_{j=1}^{\bar{x}} ls_j}{S_t}; \quad \bar{x} \in no\,of\ \ jobs\,in\,J_{dbdps} \tag{8}$$

After computing TRP, the jobs are passed to the second level of the PJMC. At this level, the job that is accepted by comparing the TRP of the job and deadline on the job, is represented as J_{s-PJMC} and is expressed as shown in equation (9).

$$J_{s-PJMC} = \begin{cases} 1\,;\, if\ \ TRP \le d_j \\ 0\,;\, otherwise \end{cases} \quad \mid \forall_j \in \left(1, \overline{N}\right) in\ J_{fq} \tag{9}$$

The accepted jobs $\left(\overline{\overline{N}}\right)$ are stored in the queue $\left(J_{sq}\right)$. The jobs are accepted based on the current workload of the system. The accepted jobs are submitted to the third level of PJMC. At this stage, the jobs are considered as a stochastic process. The accepted jobs $\left(\overline{\overline{N}}\right)$ follow a Poisson process for a particular time interval for processing the batch jobs. The total number of accepted jobs is calculated as shown in equation (10).

$$P_x\left(\overline{\overline{N}}\right) = \frac{e^{-\lambda t}(\lambda t)^{\overline{\overline{N}}}}{\overline{\overline{N}}!}\,;\ where\ \overline{\overline{N}} = 1,2,... \tag{10}$$

'λ' and 't' stand for the arrival rate of the jobs and time interval respectively. $\overline{\overline{N}}$ represents number of times an event occurs (i.e. jobs). The mean arrival rate of jobs is represented as λ' and calculated as shown in equation (11).

$$\lambda' = \frac{\sum_{j=1}^{\overline{\overline{N}}} ls_j}{t} \tag{11}$$

The jobs are accepted only if $\lambda' \le S_t$, otherwise the arrival rate of the jobs will be slowed down. The utilization of the data center is represented as ρ and is carried out as shown in equation (12).

$$\rho = \frac{\sum_{i=1}^{\overline{\overline{N}}} \frac{ls_i}{d_i}}{S_t} ;\ j \in J_{sq} \tag{12}$$

ρ denotes a ratio between mean arrival rate and service rate of jobs. It is used for avoiding traffic congestion, delay of job execution, overloading of VM and providing an efficient service for jobs submitted by users. In order to avoid overloading, the jobs are accepted only if $\rho \leqslant 1$ otherwise the jobs are forwarded to another service provider as long as $\rho > 1$. The quality and efficiency of the system are evaluated based on ρ. The 'x' jobs approved from $\overline{\overline{N}}$ are stored in J_d that is forwarded to DBDPS.

4.2 Deadline Based Dynamic Priority Scheduler (DBDPS)

The jobs that satisfy the multiple criteria of PJMC are approved and passed to DBDPS. In DBDPS, the priority value of the jobs stored in the J_d queue is calculated based on different parameters like length of the submitted job, deadline of the job, waiting time of already accepted job and the maximum computational speed of VM. Some jobs may already exist in the J_{dbdps} before submitting a new job. The minimum waiting time for processing the current job is represented as W^t. It is computed based on different parameters like length of the job, TRP and processing speed of the VM as shown in equation (13).

$$W_j^t = TRP + \frac{\sum_{j-1} ls_j}{S_t} | \forall_j \in J_d \tag{13}$$

In the above equation, W^t is calculated by adding the time required for processing the accepted jobs in the queue (i.e. J_{dbdps}) along with the processing time for accepted jobs in the queue (i.e. J_d). The jobs in J_d are processed based on their priority value. The priority value is denoted as P_d and is calculated as shown in equation (14).

$$P_d^j = \left(\frac{\left(\frac{ls_j}{d_j - W_j^t} \right)}{Max_s} \right)^{-1} ;\forall_j \in J_d \tag{14}$$

The VM with maximum processing speed is taken into the account instead of checking with every VM because, if the VM with maximum processing speed cannot complete the job within deadline, then no other VM is capable to complete the jobs within deadline. The jobs $(\bar{x})$ are sorted and stored in a queue (J_{dbdps}) based on their priority value. The job with the lowest priority value is given the highest preference and it remains in the head of the queue. The priority of the submitted jobs is dynamically varied in order to evade starvation (i.e. a user

request remains waiting for a long time to obtain resources). The prioritized jobs are passed to the CMJVM.

4.3 Contest Mapping of Jobs with Virtual Machine (CMJVM)

The prioritized jobs are dynamically mapped with the VM based on their normalized value (nv) . The normalized value is computed based on different parameters like length of submitted job, deadline of the job, waiting time of the job and communication time. There is a job in job queue (i.e. J_{dbdps}) which can utilize the processing speed of any VM as shown in equation (15).

$$\exists_j \forall_{VM} \exists_{PS} : \left[T(j, PS) \wedge Q(PS, VM) \right] \tag{15}$$

$T(j, PS)$: job utilizes the processing speed and $Q(PS, VM :)$ processing speed of the every VM. The waiting time is calculated for all existing VM present in VM supervisor The waiting time for already accepted jobs in the VM queue is represented as W_t and calculated as shown in the equation (16).

$$W_t^i = \left\{ \left(\frac{\sum_{j=1}^{m} ls_j}{S_1} | S_1 \subseteq S_e \,\&\, \forall_{VM} \right) \vee \left(\left(\frac{\sum_{j=1}^{m} ls_j}{P_k} \right) | \forall_{P_k} \in S_u \,; k \in (1, \beta) \right) where \begin{array}{l} j \in J_{dbdps} \\ i \in (1, \bar{y}) \end{array} \right\} \tag{16}$$

where m represents the number of jobs in the corresponding VM queue. VM supervisor contains all the relevant information about the VM like the total number of jobs present in queue, number of VM, processing speed of VM and the bandwidth of VM. The total number of VM is represented as $\bar{y}$. W_t is computed and stored in the matrix as $1 \times \bar{y}$ as given in equation (17).

$$W_t^i = \left[VM_1, VM_2, ..., VM_{\bar{y}} \right]; i \in (1, \bar{y}) \tag{17}$$

The waiting time is computed for the VM existing in either homogeneous or heterogeneous environment. In the proposed work, all the VMs are connected with high bandwidth. The notation 'BW' denotes bandwidth of the VM. The job can execute in any VM as given in equation (18).

$$\forall_{job,VM} P(job, VM) : \textit{job execute in VM} \tag{18}$$

The time taken for communicating a job to underlying VM is represented as C_c and it is calculated as shown in equation (19).

$$C_c^{ij} = \left[\frac{ls_j}{BW\_VM_i} \right] | \forall_j \in J_{dbdps}, i \in (1, \bar{y}) \tag{19}$$

where BW_VM represents the processing speed of the VM. C_c is required for transferring a job from the queue to the VM. It is computed for each job with all VM. It varies dynamically depending on the job length and VM bandwidth. The computed results are stored in the matrix $\bar{y} \times \bar{x}$. The total waiting time of the job is represented as WT. It is computed by comparing the elements of W_t with each column of C_c as shown in equation (20).

$$WT^{ij} = \max\left(W_t^i, C_c^{ij}\right); \forall_{job,VM} \tag{20}$$

$$\begin{array}{c} \\ VM_1 \\ VM_2 \\ \vdots \\ VM_{\bar{y}} \end{array}\begin{array}{c} \begin{array}{cccc} j_1 & j_2 & \cdots & j_{\bar{x}} \end{array} \\ \begin{bmatrix} WT^{11} & WT^{12} & \cdots & WT^{1\bar{x}} \\ WT^{21} & WT^{22} & \cdots & WT^{2\bar{x}} \\ \vdots & \vdots & \cdots & \vdots \\ WT^{\bar{y}1} & WT^{\bar{y}2} & \cdots & WT^{\bar{y}\bar{x}} \end{bmatrix} \end{array} = \max\left(\begin{array}{c} \begin{array}{cccc} VM_1 & VM_2 & \cdots & VM_{\bar{y}} \end{array} \\ \begin{bmatrix} W_t^1 & W_t^2 & \cdots & W_t^{\bar{y}} \end{bmatrix} \end{array}, \begin{array}{c} \\ VM_1 \\ VM_2 \\ \vdots \\ VM_{\bar{y}} \end{array}\begin{array}{c} \begin{array}{cccc} j_1 & j_2 & \cdots & j_{\bar{x}} \end{array} \\ \begin{bmatrix} C_c^{11} & C_c^{12} & \cdots & C_c^{1\bar{x}} \\ C_c^{21} & C_c^{22} & \cdots & C_c^{2\bar{x}} \\ \vdots & \vdots & \cdots & \vdots \\ C_c^{\bar{y}1} & C_c^{\bar{y}2} & \cdots & C_c^{\bar{y}\bar{x}} \end{bmatrix} \end{array} \right)$$

The jobs are assigned to the VMs depend on the EPS and hence it completes within its corresponding deadline. The minimum processing speed necessary for the submitted jobs is represented as E_{eps} as shown in equation (21).

$$E_{eps}^{ij} = \frac{ls_j}{d_j - WT^{ij}}; \forall_{job,VM} \tag{21}$$

The VMs may exist either in a homogeneous or heterogeneous environment depending on the data center allocation policy. The jobs are scheduled based on the normalized value for completing their execution within deadline and also balancing the system load. The job is apportioned to the VM having the potential to complete within its deadline based on the normalized value. MIPS_VM represents the processing speed of the VM. The normalized value in a homogeneous environment is represented as nv_e and it is calculated as shown in equation (22).

$$nv_e^{ij} \leftrightarrow \left[\min\left(\left[\frac{\sum_{k=1}^{m} \frac{ls_k}{d_k}}{MIPS\_VM_i} + \frac{E_{eps}^{ij}}{MIPS\_VM_i} \right] \forall_i \right) \forall_j \right]; \; where \; i \in (1, \bar{y}) \; \& \; j \in (1, \bar{x}) \tag{22}$$

Initially, the accessible processing speed of VM in a heterogeneous environment is represented as nv_{ue} and is calculated for each job with all VM as shown in equation (23). Among these VMs, the non-capable VMs are neglected $\left(i.e.if \; nv_{ue}^{ij} > 1\right)$. The potential VMs are filtered and stored in n_m as shown in equation (23).

$$n_m \xleftarrow{filter} \left\{ \left[nv_{ue}^{ij} = \max \left(\left[\frac{\sum_{k=1}^{m} \frac{ls_k}{d_k}}{MIPS\_VM_i} + \frac{E_{eps}^{ij}}{MIPS\_VM_i} \right] \forall_i \right) if \ \forall_{nv_{ue}^{ij}} \le 1 \right] \forall_j \right\}; where \begin{matrix} i \in (1, \bar{y}) \\ j \in (1, \bar{x}) \end{matrix} \quad (23)$$

Among n_m, the job is mapped to the VM having large normalized value for increasing the efficiency of the system. The jobs are assigned to the VM under the heterogeneous environment. The job is mapped to execute in a particular VM. The normalized value in a heterogeneous environment is represented as nv_u as shown in equation (24).

$$nv_u^{ij} \mapsto \begin{cases} \max(n_m) & ; if \ n_m \neq \phi \\ \min(nv_{ue}^{ij}) & ; otherwise \end{cases} \quad (24)$$

The normalized value is recalculated for already allocated VM to check whether it is capable of completing the successive job within its deadline. The job assigned to the VM depends on the three different attribute like the required processing speed of accepted job, required processing speed of incoming job and inter-arrival time between jobs. The normalized value of already allocated job in VM is denoted as nn_{VM} and it is illustrated in equation (25).

$$nn_{VM} = nv_{ue}^{(j+1)} - T \, ; if \ j \in J_{dbdps} \quad (25)$$

where T represents the inter arrival time between the job 'j' and 'j+1'. The consecutive jobs are assigned to the VM as given below

$$nv_u^{ij} \mapsto \begin{cases} nn_{VM} & ; if \ nn_{VM} < 1 \\ \max(n_m) & ; if \ (nn_{VM} < 1) \& (n_m \neq \phi); \forall_j \in J_{dbdps} \\ \min(nv_{ue}^{ij}); otherwise \end{cases} \quad (26)$$

After the completion of job, the status of the VM is updated in VM supervisor. Hence, the normalized value is recalculated for the particular VM. The recalculated normalized value for the particular VM is represented as n_r and is calculated as shown below.

$$n_r = \frac{\sum_{j=1}^{m} \frac{ls_j}{d_j}}{MIPS\_VM_i} + \frac{E_{eps}^{ij}}{MIPS\_VM_i} \mid a \ \ job \ completes \ in \ i^{th} \ VM \ ; \ j \in J_{dbdps} \quad (27)$$

After computing the n_r, the VM supervisor will check whether the VM is capable of executing the job within its deadline $(i.e. if \ n_r < 1)$. Therefore, the VM will append to the

n_m (n_m contains the eligible VM). The jobs are licensed to execute in a particular VM (i.e. specific time period) are passed to the VM Scheduler.

4.4 Virtual Machine Scheduler

The VM scheduler will define the policies during the creation of the VM. The VMs are treated as a Foreground VM (FVM) and Background VM (BVM) in the two-tier architecture. The FVM and BVM are treated as high and low priority respectively. The FVM and BVM belong to the same processing element. Hence, the full processing speed is initially allotted to the FVM. The processing speed of BVM fully depends on the processing speed of FVM. The FVM and BVM are complements of each other with respect to their processing speed. The processing speed of the VM is denoted as PS. The relation between FVM and BVM is given in the equation (28).

$$PS_{FVM} \propto \frac{1}{PS_{BVM}} \tag{28}$$

The VM scheduler contains utilization manager and VM switcher components. The utilization manager monitors the utilization rate of both FVM and BVM. Based on the utilization rate, the VM switcher schedules the successive jobs between FVM and BVM by issuing licenses. The jobs are stored in the particular VM queue (VM_q) after issuing the license. Initially, the jobs from the VM_q are allotted to the FVM for their execution. Due to external delay, communication delay and synchronization delay, the job may not utilize the full processing speed of the VM. The processing speed allocated for FVM is represented as PS_{FVM} and it is defined in equation (29).

$$PS_{FVM} = \frac{(util(FVM) * MIPS)}{100} \tag{29}$$

The job in VM queue (VM_q) is allocated to the FVM. The job does not utilize the full processing speed of the VM during its execution. Therefore, the utilization of FVM decreases below the threshold value (96%). So, the remaining processing power of FVM can be allocated to the BVM dynamically. The VM switcher will schedule the subsequent job to BVM to increase the resource utilization. The processing speed of the BVM is represented as PS_{BVM} and it is calculated dynamically as shown in equation (30).

$$PS_{BVM} = \frac{((1 - util(FVM)) * MIPS)}{100} \tag{30}$$

The jobs in corresponding VM_q are concurrently executed in the VM. The jobs in each VM_q are allocated between FVM and BVM using VM switching algorithm. In the VM switching algorithm, the jobs are scheduled between FVM and BVM depending on the utilization rate. After the completion of FVM job, the job running in BVM migrates to FVM for maintaining the preference for higher priority jobs. During migration, the status of running job in BVM is handed over to the FVM.

VM Switching Algorithm

(1) Get a job 'j' from the J_{dbdps}

(2) **for** each j in J_{dbdps} mapped with VM **do**

(3) ‖ly compute $\forall_{VM}$, j in VM_q

(4) **if** $VM_j = \phi; \forall_{VM}$ *then* /*VM_j represents job execute in VM */

(5) $j \in VM_q \xrightarrow{assign} FVM$

(6) **else if** $PS_{FVM} < 0.96$ *and* j_i *process in FVM then* /*to improve utilization*/

dynamically allocate PS_{BVM} ; $j_{i+1} \xrightarrow{assign} BVM$

(7) **if** j_i complete in FVM **then**

(8) $j_{i+1} \xrightarrow{migrate} FVM$; $j_{i+2} \xrightarrow{migrate} BVM$

/* move the job from BVM to FVM to maintain priority*/

(9) else $j_{i+2} \xrightarrow{assign} BVM$

(10) **end if**

(11) **end if**

(12) $j_{i+1} \xrightarrow{assign} FVM$

(13) **end if**

(14) **end for**

(15) **Goto** (1) for scheduling subsequent jobs;

The VM switching algorithm will effective schedule the jobs between the foreground and background VM for effective processing speed utilization of the VM.

4. Simulation and Results Analysis

CloudSim, used as a simulation tool, provides basic classes that describe data center, virtual machine, computational resources and policies for scheduling and provisioning of resources explained by R. Buyya [27]. The utilization of the cloud has been enhanced with the development of new strategies with different policies, scheduling algorithms, mapping and load balancing. Hence, an Adaptive Multilevel Scheduling System (AMSS) was developed for increasing the system performance and the resource utilization. The performance of the AMSS and existing algorithms are evaluated using CloudSim. The computational power of the data center is considered as 5750 MIPS given by Zhuge Bin, et al. [28]. The number of jobs, number of VMs and the simulation parameters is shown in **Table 1**.

Table 1. Simulation parameters

Parameter	Range
Job Size (MI)	5,000-25,000
Number of jobs	8-100
Computing Power on Data Center (MIPS)	5750
Computing Power of VM in homogeneous environment (MIPS)	500
Computing Power of VM in heterogeneous environment (MIPS)	1000-2500
Number of VM	10-15
Deadline (ms)	10-100

5.1 Experimental environment and result analysis

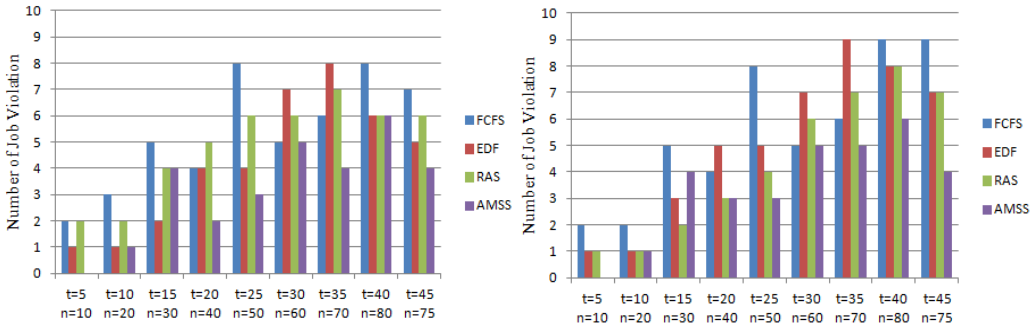
In this simulation, CloudSim creates a VM in either homogeneous or heterogeneous environment. The jobs are submitted to the VM for their execution. The incoming jobs are considered as a Poisson process because the jobs are submitted in a specific time interval. The jobs are submitted and tested in both homogeneous and heterogeneous environment of the CloudSim simulation toolkit. The bandwidth of the VMs varies with respect to the system architecture.

5.2 Performance analysis

In the proposed system, job submitted at the time t=0 become zero, so the waiting time of already accepted jobs and $j \in J_{dbdps}$ are considered as zero. The number of jobs is linearly increased up to $\rho \leq 1$. The arrival jobs are slowed down only if $\rho \geq 1$. The efficiency of the system is achieved using J_{s-PJMC} and the jobs will be completed within their specified deadline having minimum variance. The jobs are preprocessed with different constraints to improve the user satisfaction. The jobs are prioritized with different parameters. This system will effectively schedule the jobs based on their priority value and also balance the load using CMJVM. The proposed work can efficiently schedule the jobs in the underlying resources. Moreover, the delay of currently running job does not affect the subsequent jobs due to the introduction of VM Scheduler in the two-tier architecture. The scalability can be achieved by comparing the total processing speed of the VM and arrival rate of the jobs. The proposed work dynamically balances the load among the available VMs.

5.2.1 Job Violations

The proposed work focuses to reduce the average number of job violating its deadline and thereby increase user satisfaction through the completion of job within its deadline. Initially at t=5, the submitted jobs (n=8) are less than available VM so $j \in J_{dbdps}$ becomes ϕ and waiting time of the job becomes zero. Number of jobs submitted to the VM will increase linearly up to $\rho \leq 1$. At t=40, $\rho \geq 1$ will slowed down the arrival rate of jobs.



(a). Job Violation in Homogeneous Environment

(b). Job Violation in Homogeneous Environment

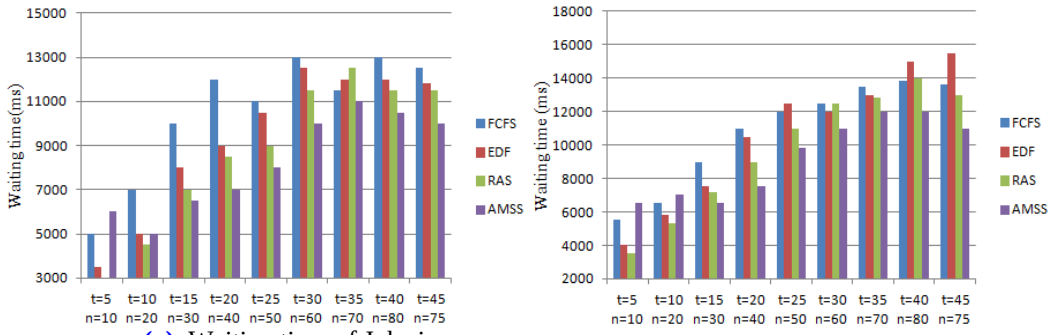
Fig. 2. Impact Job Violations using AMSS

Fig. 2 shows the impact of job violating using AMSS. In **Fig. 2**, **(a)** and **(b)** represent the number of jobs violating their deadline in the homogeneous and heterogeneous environment respectively. The jobs are filtered in AMSS only if the job requires more processing speed than

the available processing speed of the VM. Because, those jobs would degrade the system performance. Among the accepted jobs, the delay of current running job may affect the subsequent jobs. But, by the deployment of VM scheduler in the two-tier VM architecture will effectively schedule and complete the jobs within their deadline. **Fig. 2** shows only a small number of jobs violating their deadline in AMSS compared with other algorithms like FCFS, EDF and RAS since the jobs are preprocessed in AMSS. Among the accepted jobs, some of them may violate their policy with minimum deviation from their deadline compared to other algorithms. The minimum deviation is due to the communication delay and previous workload of the VM. The job is assigned to the VM that exist in either homogeneous or heterogeneous environment.

5.2.2 Waiting time

The waiting time of the job decreases in AMSS compared with other algorithms (i.e. FCFS, EDF, HLBA and RAS) due to the introduction of VM scheduler in the two-tier VM architecture. Here, the processing speed of the VM has been shared among the foreground and background VM. The delay of currently running job does not affect the remaining jobs present in the queue because the consecutive job in the VM_q will effectively utilize the remaining processing speed of the VM using the background VM.



(a). Waiting time of Jobs in Homogeneous Environment

(b). Waiting time of Jobs in Homogeneous Environment

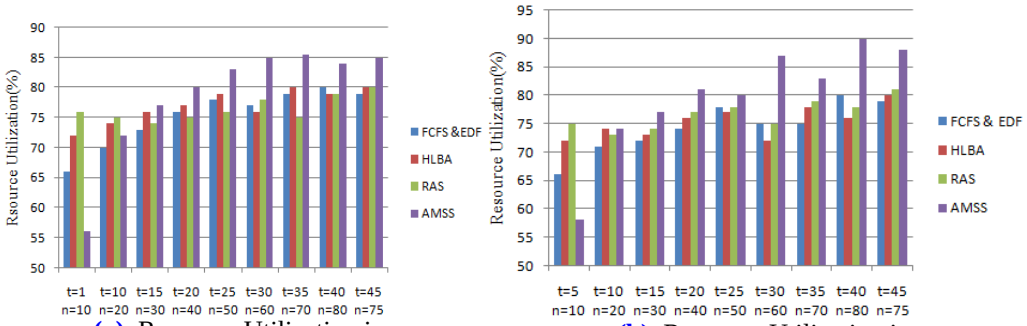
Fig. 3. Impact of Jobs Waiting Time using AMSS

Fig. 3 shows that the waiting time of jobs is reduced using AMSS. In **Fig. 3, (a)** and **(b)** represents the waiting time in a homogeneous and heterogeneous environment respectively. The AMSS minimizes the user waiting time and achieves certain fairness among the resources in the homogeneous environment. The jobs are mapped with the appropriate VM to minimize waiting time in a heterogeneous environment. Moreover, the processing speed of VMs is fully utilized in AMSS due to the deployment of VM scheduler in the two-tier VM architecture compared with other algorithms and thereby decrease the waiting time of the job. The VM scheduler is introduced for effectively scheduling the jobs between FVM and BVM. Hence, the waiting time of the job is decreased comparing to other algorithms.

5.2.3 Resource Utilization

The VM switching algorithm in the proposed AMSS decreases the resource utilization initially compared with other algorithms like FCFS, EDF, HLBA and RAS if $jobs \leq VMs$ because the jobs are preprocessed and filtered. The resource utilization is increased gradually when the number of incoming jobs increases linearly. In HLBA, the jobs are scheduled to the VM

having high computational power. Therefore, VM having low computational power remains idle. **Fig. 4** shows the impact of resource utilization using AMSS. In **Fig. 4**, **(a)** and **(b)** represents the resource utilization in homogeneous and heterogeneous environment respectively. Since the VM exist in the homogeneous having idle computational speed and their resource utilization is upto 86% (i.e. processing speed of VM is 1000 MIPS so that 750 MIPS remains idle). But, the resource utilization of VM is up to 90% in a heterogeneous environment due to the characteristics of job. Moreover, the processing speed of VMs is fully utilized in AMSS due to the deployment of VM scheduler in the two-tier VM architecture compared with other algorithms and thereby increases the VM throughput. The VM scheduler is introduced for scheduling the jobs effectively between FVM and BVM. Hence, the resource utilization is increased to the range of 4% to 6% compared with other algorithms.



(a). Resource Utilization in homogeneous environment

(b). Resource Utilization in heterogeneous environment

Fig. 4. Impact of Resource Utilization using AMSS

5. Conclusion and Future Works

The proposed research work examined the difficulties of dynamic batch job scheduling. The objectives of this work were to bolster the user satisfaction, to mitigate job violating its policy and to maximize resource utilization. To achieve these objectives, AMSS has been proposed for scheduling the batch jobs. The user satisfaction was achieved by neglecting the job that doesn't satisfy the criteria of PJMC. The number of jobs violating their deadline was reduced by filtering the jobs using multiple criteria. The priority was dynamically assigned to the accepted jobs in DBDPS in order to avoid starvation. The prioritized jobs were efficiently mapped with VM either in homogeneous or in the heterogeneous environment using CMJVM and thereby efficiently balanced the load. The VM Scheduler has been deployed in the two-tier VM for effectively scheduling the jobs between FVM and BVM. Hence, utilization of resources was increased to the ranges of 4% to 6%.

AMSS outperforms the existing scheduling algorithms by reducing the number of jobs violating their deadline that improves the user satisfaction. It also focused on load balancing that increases throughput and also resource utilization. In future, the work can be extended to develop an efficient cost and energy aware scheduler for processing both dependent and independent jobs.

References

[1] M. D. Dikaiakos, D. Katsaros, et al., "Cloud Computing: Distributed Internet Computing for IT and Scientific Research," *IEEE Internet Computing*, vol. 1, no. 5, pp. 10-13, 2009. Article (CrossRef Link)

[2] K. Dinesh, G. Poornima and K. Kiruthika, "Efficient Resources Allocation for different Jobs," *International Journal of Computer Application*, vol. 56, no. 10, pp. 30-35, 2012. Article (CrossRef Link)

[3] P. Mell, T. Grance, "The NIST Definition of Cloud Computing," *NIST Special publication*, pp. 800-145, 2011.

[4] Md. Sabbir Hasan, E. N. Huh, "Heuristic based Energy-aware Resource Allocation by Dynamic Consolidation of Virtual Machines in Cloud Data Center," *KSII Transactions on Internet and Information Systems*, vol. 7, no. 8, pp. 1825-1842, 2013. Article (CrossRef Link)

[5] D. Jung, T. Suh, H. Yu and J. M. Gil, "A Workflow Scheduling Technique Using Genetic Algorithm in Spot Instance-Based Cloud," *KSII Transactions on Internet and Information Systems*, vol. 8, no. 9, pp. 3126-3145, 2014. Article (CrossRef Link)

[6] K. M. Sim, "Agent-Based Cloud Computing," *IEEE Transactions on Services Computing*, vol. 5, no.4, pp.564-577, 2012. Article (CrossRef Link)

[7] Li. Chunxiao, A. Raghunathan and Niraj K. Jha, "A Trusted Virtual Machine in an Untrusted Management Environment", *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 472-483, 2012. Article (CrossRef Link)

[8] Y. Zhang, H. Franke, et al., "An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 3, pp. 236-247, 2003. Article (CrossRef Link)

[9] D. Carrera, M. Steinder, et al., "Autonomic Placement of Mixed Batch and Transactional Workloads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 2, pp. 219-231, 2012. Article (CrossRef Link)

[10] A. Silberschatz, P. B. Galvin and G. Gagne, "Operating System Concepts," *ISSN-978-1-118-06333-0, 9th edition*, 2011.

[11] D. Nurmi, R. Wolski and J. Bervik, "Probabilistic Reservation Services for Large-Scale Batch-Scheduled Systems," *IEEE Systems Journal*, vol. 3, no. 1, pp. 6-24, 2009. Article (CrossRef Link)

[12] A. W. Mu'alem and D.G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529-543, 2001. Article (CrossRef Link)

[13] V. Gamini Abhaya, Z. Tari, et al., "Performance Analysis of EDF Scheduling in a Multi-Priority Preemptive M/G/1 Queue," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2149-2158, 2014. Article (CrossRef Link)

[14] S. Abrishami, M. Naghibzadeh, et al., "Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 158-169, 2013. Article (CrossRef Link)

[15] M. Stillwell, F. Vivien and H. casanova, "Dynamic Fractional Resource Scheduling versus Batch Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 521-529, 2012. Article (CrossRef Link)

[16] R. Baraglia, G. Capannini, et al., "A multi-criteria job scheduling framework for large computing farms," *Journal of Computer and System Sciences*, vol. 79, no. 22, pp. 230-244, 2013. Article (CrossRef Link)

[17] B. Xu, C. Zhao, et al., "Job Scheduling algorithm based on Berger model in cloud environment," *Advances in Engineering Software*, vol. 42, no. 7, pp. 419-425, 2011. Article (CrossRef Link)

[18] E. U. Munir, J. Z. Li, et al., "A new heuristic for task scheduling in heterogeneous computing environment," *Journal of Zhejiang University SCIENCE A*, vol. 9, no. 12, pp. 1715-1723, 2008. Article (CrossRef Link)

[19] Y. H. Lee, S. Leu, et al., "Improving job Scheduling algorithm in a grid environment," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 991-998, 2011. Article (CrossRef Link)
[20] Y. Zhao, L. Chen, et al., "Efficient task scheduling for Many Task Computing with resource attribute selection," *China Communications*, vol. 11, no. 12, pp. 125-140, 2014. Article (CrossRef Link)
[21] L. Zhou and H. Wang, "Toward Blind Scheduling in Mobile Media Cloud: Fairness, Simplicity, and Asymptotic Optimality," *IEEE Transaction on Multimedia*, vol. 15, no. 4, pp. 735-746, 2013. Article (CrossRef Link)
[22] L. Zhou, Z. Yang, et al., "Exploring blind online scheduling for mobile cloud multimedia services," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 54-61, 2013. Article (CrossRef Link)
[23] X. Liu, C. Wang, et al., "Backfilling under Two-tier Virtual Machines," *IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 514-522, 2012. Article (CrossRef Link)
[24] X. Liu, C. Wang, et al., "Priority-Based Consolidation of Parallel Workloads in the Cloud," *IEEE Transaction on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1874-1883, 2013. Article (CrossRef Link)
[25] C. Zhao, S. Zhang, et al., "Independent Tasks Scheduling Based on Genetic Algorithm in Cloud Computing," *International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1-4, 2009. Article (CrossRef Link)
[26] M. Kuanr, P. Mohanty, S. C. Moharana, "Grouping-Based Job Scheduling in Cloud computing using Ant Colony Framework," *International Journal of Engineering Research and Applications*, 2013.
[27] R. Buyya, R. Ranjan and R. N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities," *International Conference on High Performance Computing & Simulation, HPCS '09*, pp. 1-11, 2009. Article (CrossRef Link)
[28] Z. Bin, D. Li, et al., "Resource scheduling algorithm and ecnomic model in ForCES networks," *China Communications*,vol. 11, no. 3, pp. 91-103, 2014. Article (CrossRef Link)

Dinesh Komarasamy received the Bachelor of Engineering and Master of Engineering in Computer Science and Engineering from Anna University, Chennai, India in 2010 and 2012 respectively. He is currently pursuing Ph.D. at College of Engineering, Anna University, Chennai, India. His research area is focused on job scheduling in cloud computing.

Muthuswamy Vijayalakshmi is an Assistant Professor in the Department of Information Science and Technology, College of Engineering, Anna University, Chennai, India. She has 12 years of teaching experience. She received the Bachelor of Engineering in Instrumentation and Control Engineering from National Institute of Technology, Trichy in 1998 and Master of Engineering in Computer Science and Engineering from College of Engineering, Anna University, Chennai, India in 2002. She completed her Ph.D in Mobile Databases from Anna University, Chennai, India in 2009. Her research areas include mobile databases and mobile cloud computing.