A Multi-Level Scheduling for Resource Provisioning Mechanism in Cloud Systems

Mohd Hairy Mohamaddiah, Azizol Abdullah, Masnida Hussin, Shamala Subramaniam
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia.
azizol@upm.edu.my

***Abstract*—Cloud computing has emerged as one of the paradigm in supplying compute resources to the users. It is capable to support heterogeneous applications demands and requirements for its job processing. Hence, agility of demands for job processing from the clients often affects the resource states, resulting to over or under provision resources state. This will impact the cloud provider's performance in executing the required jobs within the shortest amount of time. In this paper, we address the over and under provision of resources to execute the heterogeneous jobs within shortest time possible. We proposed a multi-level scheduling for provisioning mechanism by incorporating job ranking mechanism and best match resource allocation. Our simulation results show that our mechanism achieves better execution time compared to other scheduling mechanisms**

***Index Terms*—Provisioning Mechanism; Scheduling; Allocation; Multi-Level Scheduling.**

I. Introduction

Cloud offerings such as compute elements, platform or software as a service had diverted the norms of application processing environment. The cloud deployment services will provision servers, storage and its associated components to fulfill clients' demand. However, heterogeneity of applications complicates the resource provisioning process in fulfilling their demand. For instance, complex scientific applications which handle large scale data require a huge sum of compute resources. In addition, complex scientific jobs are often represented by a workflow consisting of a series of interdependent services [1]. However, difficulties may be faced by the execution process due to remaining jobs being processed at the providers. Moreover, unavailable resources during resource request will influence the execution time for the jobs [2-3]. Therefore, these situations require dynamic provisioning mechanisms to cater heterogeneity of jobs to be processed in the cloud.

Hence, in detail, our proposed mechanism introduced a multi-level scheduling scheme which combines scheduling at job and resource levels. Our mechanism is realized using two different procedures. The first procedure is at the job level, by determining the highest rank of jobs. The second procedure is at the resource level, where we adopted work by Li et al. [3], in finding the best fit resources based on feedback information generated by different resources in the cloud. Our work is evaluated by using discrete event simulations by varying the number of tasks and comparing it with other scheduling mechanisms. Our results show that the proposed approach minimizes job execution time compared to other scheduling mechanisms.

The remainder of this paper is organized as follows. A review of related works is presented in Section 2. In Section 3, we described the problem formulation and our system model used in the paper. Our proposed mechanism is presented in Section 4. Section 5 details our simulation settings and presented the results obtained. Finally, conclusions are made in Section 6.

II. Resource Provisioning

Resource provisioning is a broad area. According to Manvi & Shyam [4], resource provisioning can be defined as the allocation of a service provider's resources to a client. It involves releasing the requested compute resources in the midst of other resources that are simultaneously running. Hence, due to uncertainty of resources [2,5], heterogeneities and criticalities of jobs, might result to resources become fierce if the available compute resources are not enough to process the jobs. In resource provisioning, several mechanisms for heterogeneous jobs have been proposed in multi-environment cloud computing [2,5-8] to provide a better job execution time in uncertainty status of resources (i.e. over or underutilized resources). Work by Babu & Krishna [6] resolved a problem of over and under loaded resources for processing tasks by proposing an algorithm called honey bee behavior inspired load balancing. The proposed mechanism improved the average execution time and showed reduction in waiting time for tasks on queue. Another work by Ryan & Lee [7] also improved job execution time significantly via their proposed approach; Multi-Tier Resource Allocation scheme in the data intensive applications environment. In mobile cloud, for task allocation, Hung & Huh [8], used a genetic mechanism to improve task scheduling and allocation, achieving better performance in task processing time. Other mechanisms being attempted on job level, such as job preemption mechanism proposed by Li et al. [3] and the two multi-criteria meta-heuristic algorithms proposed by Moschakis & Karatza [9] had also improved performance and job execution time. While these works treated resource allocations mechanisms between job and resources separately, our mechanism deals with both job and resources simultaneously.

III. Problem Formulation And System Model

In this section we will elaborate on the problem formulation by defining our application model which comprises of application and computation resources representations. We also define our system model to introduce our proposed mechanism.

A. *Problem Formulation*

In this section, we will define the terms used in this paper. Then, we will formulate our problems based on several preconditions.

Definition 1: Our applications denoted as A_i consists of numbers of jobs J_i and each job consists of number of tasks t_i;

$$t_i \in j_i \ \in A_i \tag{1}$$

Our target applications are complex applications, which can be decomposed into jobs and tasks as lowest levels, such that, each task is interdependent of each other [10]. Therefore, our strategy in the proposed mechanism is to execute a given a set of jobs to set of machines with their uncertainty availability. We also imposed job allocation mechanism without any job preemption in a minimum execution time. We modeled our application based on Definition 2;

Definition 2: The pool of tasks decomposed from the jobs will be represented as directed acyclic graph (DAG), G=(V,E,w,c) model with its precedence relations, where V={V_1,V_2,…,V_n}. V is tally with their tasks and the directed edge e_{ij}; (i,j) ∈ E presents the communication between subtasks V_i and V_j, $w(v_i)$ associated with task v_i ∈ V represents its computation time and $c(e_{ij})$ represents its communication time between task V_i and task V_j with a corresponding transferred data, $d(e_{ij})$, if and only if V_i is completed before V_j. Every job has its different tasks with different task weights. A task with no predecessor is called an entry task, V_{entry}, whereas an exit task, V_{exit}, is one that does not have any successor [11]. In our implementation that will be discussed in the next section, categorize the jobs into two different modes; Advanced Reservation (AR) mode and Best Effort (BE) mode. The AR mode is where resources for the jobs are reserved in advanced. While for the BE mode, resources will be provisioned the soonest based on resources availability.

Definition 3: Let R={R_1,R_2,$R_{3,…,}$$R_n$} is set of pool resources from multi-cloud providers. These resources which are virtual machines in the private cloud will be represented as a resource pool graph. The topology of the resources is denoted as R=(C,D). C is set of vertices which represents the resource nodes where d_{ij} ∈ D is directed link between resources node [8]. Each resource has its own processing rate and bandwidth.

The generated DAG model will display the dependency between each task in a job. In our proposed mechanism there is no job-preemption when the job is scheduled to be executed even though we have an AR mode type of jobs. Our proposed job prioritized mechanism will assist in making decision for the best jobs to be processed without decrementing the criticality of job under the AR mode.

B. *System Model*

Our focus for this work is on Infrastructure-as-a-service (IaaS) clouds. We can say that our system model setup in Figure 1 is similar to the interconnecting grid platform setup [11,12].

We introduced two service provisioning managers; Distributed Virtual Environment (DVi) manager and Resource Infrastructure Manager (RiM). The DVi manager is responsible for the allocation of designated resources requested, while RiM manages local and outsource resources. The information sent and received between DVi and RiM is crucial in order to avoid scenarios such as over-provisioning or under-provisioning of resources.

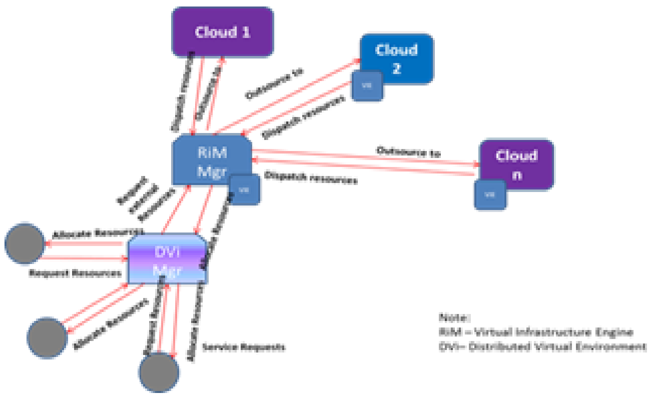

Figure 1: System model

Each cloud managed by RiM and DVi are interconnected via a high speed link. The bandwidth and intercommunication are assumed to vary. We also assumed that message between both managers can be transmitted, meaning that there is a communication route between them via the high speed link. Each cloud contains set of machines, m with identical resources (virtual machines), r and each resource has their own associated processing capacity. We adopt the processing capacity model Pr by Hussin & Latip [14]. We will use the calculated values from the adopted model in our allocation mechanism.

IV. Resource Allocation For Provisioning

One of the major challenges for job allocation is on how to schedule complex and computing intensive applications faster and more efficiently. Furthermore, with the ever growing demands for these applications, it is crucial to provide fit resources from wide variety of processing power provided by the cloud providers. In this section, we begin with the description on how we model the application, and followed by the proposed mechanism for job allocation.

A. *Application Model*

As mention in 3.1, complex and computation intensive applications come from formation of different jobs. Each job has pool of tasks. First, our application model will be based on job processing requirement, job ranking and scheduling. The jobs model is part of an allocation process which is solely managed by RiM and DVi. The processing requirement of a task in a job, as defined in Equation (2), will depict the task sizes.

$$\mathrm{O_n} = \sum_{i=1}^{n} \frac{S_i}{p} \tag{2}$$

where S_i: Size Task of Job Vi
p: Processing capacity of a referred processor node

Our assumption for this model is that the processing requirement must be less then available resource capacity.

B. *Proposed Mechanism*

Basically, our mechanism is based on job prioritization and cloud resource allocation scheme. The job prioritizing stage is the stage where we identify and rank the jobs based on our multi-condition priority mechanism. We had adopted work by Li et al. [3] for the cloud resource allocation scheme, where the cloud selection is based on cloud resource information status. We then execute the job by applying both mechanisms.

i. Job Prioritizing Stage

During the request stage, the cloud subscribers will submit their requests for computation resources as a service request. Then, these jobs will be transformed into job requests. This approach will then be formulated into a DAG model. This is where our proposed DVi and RiM managers function. This subsection will explain our two-level prioritization stage for the proposed mechanism.

Stage 1: Identifying the priority list

This stage comprises of the generation of priority task list from the jobs and identification of the group of critical path tasks from the defined priority lists. To generate the job priority task, we first utilized the same technique described by Islam et al. [15] (which will provide a feasible schedule for the jobs). This technique will come out with the job deadlines. The generated deadlines will be treated as job priority criteria. To generate job priority, we will first perform experiments without the cloud resources by using easy backfilling algorithm. Then, the job priority, Dd, is calculated based on the Equation (3):

$$D_d = \begin{cases} \mathrm{st_i} + (sf\,.\,\mathrm{ta_i}) & \text{if } [\mathrm{st_i} + (sf.\mathrm{ta_i})] < exe \\ \text{exe} & \text{otherwise} \end{cases} \quad (3)$$

where $\mathrm{st_i}$ is the request of job submission time, exe is its completion time, $\mathrm{ta_i}$ is the job's turnaround time (i.e., the difference between the tasks completion and submission times), and *sf* is a stringency factor that indicates the urgency of the applications. If *sf* = 1, then the job deadline is completion under the easy backfilling scenario. We evaluate the strategies based on different stringency factors (i.e., from 0–1); the factors indicate different scenarios, such as tight, normal, and relaxed deadline scenarios [12]. This assumption will generate a realistic job priority for each job request setup. The derived values will be the first condition.

The next step is to define the job criticality, which will be derived from the critical paths of tasks from the DAG model generated. As mentioned, the jobs are heterogeneous. This value is calculated based on a set of vertices in the DAG, of which the values of EST (5) and LST (6) are equal [3]. The earliest start time EST(vi) of node Vi can be computed recursively by traversing the DAG downward starting from the entry node Ventry [8]. We will evaluate the values derived using the earliest start time and latest finish time of a task as defined in Equation (4) and (5):

$$\mathrm{EST}(v_i) = \max_{\mathrm{v}_i \in \mathrm{pred}(\mathrm{v}_j)} \{\mathrm{EST}(\mathrm{v}_j) + \mathrm{AT}(\mathrm{v}_j)\} \quad (4)$$

$$\mathrm{LST}(v_i) = \min_{\mathrm{v_j} \in \mathrm{succ}(\mathrm{v_i})} \{\mathrm{LST}\,(\mathrm{v}_j)\} - \mathrm{AT}(\mathrm{v_i})\} \quad (5)$$

The second condition is the critical path tasks of the jobs that will be computed from Equations (5) and (6). Then, from the identified tasks, we will calculate the communication/computation ratio, $\mathrm{CC_r}$, by using the equation below:

$$\frac{\sum W_{v_i}}{\sum C_{e_{ij}}} \quad (6)$$

where $\sum W_{v_i}$ is the total computation cost of the tasks of job *i*, and $\sum C_{e_{ij}}$ is the total startup communication cost of edge *i* and *j*. The calculation will determine the job granularity and classification, that is, whether the job is computationally intensive or not. A higher value indicates that the job is computationally intensive and will be assigned the highest priority; this will be our next condition for job prioritization. On the basis of the previous equations, the relationship between critical paths of tasks *i*, CPt_i in a job with the ratio can be defined as follows:

$$CPt_i = \max_{0 \le i \le 1} CC_r \quad (7)$$

Therefore, the maximum value for each job indicates that the job is critical and will have the highest priority for resource allocation.

Stage 2: Determining the job rank

During this stage, each job is assigned a preliminary job rank, Jr_u. The rank results will assist in determining the highest-priority job to be scheduled. Here, the ranks are calculated by adopting a job ranking phase mechanism applied by Yousaf & Welzl [16]. The work uses HEFT algorithm to define the job rank. In HEFT, an upward rank is generated and recursively calculated starting from the exit tasks of a job. The job rank, Jru calculation is as follows:

$$Jr_u = \begin{cases} \overline{w_i} + \max_{v_i \in succ\,(v_j)} \{\overline{w_i} + rank_u(v_j)\} \\ \overline{w_i} \quad v_{i\,=\,endtask} \end{cases} \quad (8)$$

where $\overline{w_i}$ is the average computation cost for all tasks in the job. Note that *succ* (v_j) is a set of immediate successors of v_i. The termination point of this rank is where $v_{i\, \neq endtask}$, and its value is equal to $\overline{w_i}$.

Our proposed mechanism will use the calculated values from (3–8) to form a job ranking. Jobs identified with the highest ranks in the list based on the four conditions stated above will be given utmost priority to be processed in the cloud, whereas jobs with the lowest rank will be prioritized

ii. Cloud Resources Allocation Scheme

We had adopted online adaptive scheduling introduced by Li et al. [3] for scheduling tasks to the available resources by selecting the best fit resources for the pool of tasks. The adopted mechanism is using the latest resource information state updated by the resource manager. In the resource information, the estimated job finish time, T_{jft} for a job is stored in its resource information and defined as below:

$$T_{jft} = ERAT_i + ETM_{im} + S_i/b \quad (9)$$

where $ERAT_m$ is estimated resources available time of cloud m and ETM_{im} is the execution time matrix of job *i* running on cloud m, and S_i/b is the data transfer time assuming the size of the job, *i* is S_i and *b* is the network bandwidth.

However, the estimated job finish time may not be accurate. Therefore, the mechanism we adopted from Li et al. [3] will be adjusted to hinder any delay for resource allocation and job processing. This mechanism will then re-examine the static resource allocation with the actual execution time of a finished job and a predefined feedback factor for each cloud resource, *m*. Therefore, the actual execution time of job *i* in cloud *m* is:

$$\Delta T_{jft} = T_{jft} - ERAT_i \tag{10}$$

With the feedback factor, f_{dm}:

$$fd_{im} = \alpha \times \frac{\Delta T_{jft} - S_i/b - ETM_{im}}{S_i/b + ETM_{im}} \tag{11}$$

where α is the constant value from 0 to 1. For every job completed, RiM will update the feedback factor value, and this feedback factor is copied to other cloud providers. Therefore, when the next job arrives for processing, the feedback estimated earliest finish time is calculated as:

$$Tfd_{im} = ERAT_{im} + (1 + fd_{im}) \times (S_i/b + ETM_{im}) \tag{12}$$

From the calculated value, the resources to process the jobs will be discovered based on the best fit cloud resources available time.

V. Results Analysis And Discussion

A. Simulation Setup

We developed a discrete event simulations based on Java Programming. Our system setup in the simulation is based on system model in Figure 1. In each simulation run, we simulate a set of 64 different jobs. Among these sets of Jobs, we differentiate the jobs with two types of job mode; Advanced Reservation (AR) mode and Best Effort (BE) mode as mentioned in section 2; with 52 jobs running in BE Mode and 12 jobs in AR mode. Each job composed of up to 16 tasks. There are 4 clouds in the simulation. All 64 jobs will be submitted to random clouds at a random arrival time. In the simulation, we set the parameters randomly according to the maximum and minimum values shown in Table 1. We executed 10 simulation runs with different sets of jobs.

Table 1
Range of Set Parameters

Parameters	Minimum	Maximum
ETM*ij*	25	100
Number of VMs in a Cloud	20	100
Number of CPU in a VM	1	8
Network bandwidth in a VM	2	100
Memory in a VM	32	2048
Disk space in a VM	5,000	100,000
Speed of Copy in a disk	100	1000
Number of CPU in a lease	50	100
Network bandwidth in a lease	10	800
Memory in a lease	100	10000
Disk space in a lease	500,000	20,000,000

The performance metrics chosen for our simulations is the average job execution time. It is computed by subtracting the job finish time from the job submission time [3]. We then implemented our proposed mechanism in two different scenarios: tight scenario and loose scenario. For the first scenario, we set the inter-arrival time of the AR tasks randomly; they should start not later than 30 seconds after a task arrives. For the second scenario inter-arrival times of the tasks are set close to each other. We implemented the generated DAG task graph using our proposed mechanism. In the implementation, the constant α is being used to compare between different mechanisms affecting the average job execution in a unit of time.

B. Results

Experimental results are presented in two different scenarios explained in the previous section. As a baseline for comparison, we compared our mechanism with two dynamic scheduling mechanisms, dynamic min-min scheduling (DMMS) and dynamic list scheduling (DCLS) [17]. We name our mechanism as Multi-Level Scheduling (MLS). Figure 2 shows the average execution time in the loose scenario. As shown in the figure, our mechanism has a better execution time compared to the other two dynamic scheduling mechanisms. The average execution time in MLS is, on average, about 46% faster. This performance can be explained by the two levels of scheduling mechanism applied that are able to reduce the waiting time, hence improve the execution time. Our mechanism is able to rank the jobs efficiently even though there are two different modes of jobs. It shows that no preemption needed, although the BE job mode is higher ranked compared to AR mode. Furthermore, the dynamic updating resource information in the combined mechanism helps to discover the best available resources. We believed that in loose situation, the underutilized resources are being managed efficiently, therefore improving all incoming jobs processing.

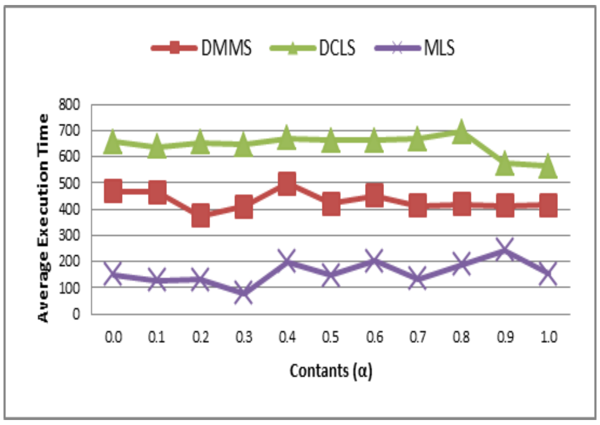

Figure 2: Average execution time with different allocation mechanisms in Loose Scenario

Meanwhile, Figure 3 shows the average execution time in the tight scenario. In this scenario, our mechanism work comparatively similar to other mechanisms. However, the execution time by using MLS slightly outperforms other mechanisms. This may due to, in tight situations, the arrival gaps are tight between jobs, and we believed that our job ranks mechanism needs a longer execution time to rank the jobs thus execute the jobs to the matched resources. The same also happens for the resources best match mechanism. The dynamic updating resource information is based on estimated finish time of previous job. Thus, in this scenario, the actual finish time is later than estimated, resulting to delay in the jobs execution time. However, our proposed mechanism has successfully tackled this issue. It showed a better execution time compared to other mechanisms because it ranks the most important jobs to be processed first.

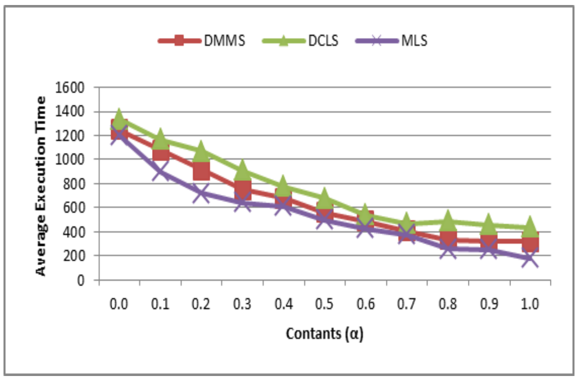

Figure 3: Average execution time with different allocation mechanism in Tight Scenario

VI. CONCLUSION

Cloud users' demands are becoming more agile and heterogeneous. This will affect the resources capability in the cloud. It will result to uncertainty of resources, either over or under provision. In this paper, we present a resource provisioning mechanism for heterogeneous applications in cloud systems. We proposed one scheduling algorithms, Multi-Level Scheduling (MLS) algorithm purposely for resource provisioning mechanism. Our mechanism combined scheduling mechanisms from two levels, jobs and resource levels. The jobs level forms job ranks and determines jobs ranking in a cloud system. Resource level determines the best matched resources based on the dynamic updated resource information. Simulations results show that the MLS outperforms DCLS and DMMS in both scenarios, tight and loose scenario. The proposed provisioning mechanism demonstrates a robust job execution regardless of different scenarios.

REFERENCES

[1] Wei,Y.., Blake, M. B. and Saleh, I. "Adaptive Resource Management for Service Workflows in Cloud Environments," *2013 IEEE Int. Symp. Parallel Distrib. Process. Work. Phd Forum*, pp. 2147–2156, 2013.

[2] Tchernykh, A., Schwiegelsohn, U., Alexandrov, V. and Talbi, E. "Towards Understanding Uncertainty in Cloud Computing Resource Provisioning," *Procedia Comput. Sci.***,** vol. 51, pp. 1772–1781, 2015.

[3] Li,J., Qiu,M., Ming,Z., Quan,G., Qin,X., and Gu,Z., "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *J. Parallel Distrib. Comput*., vol. 72, no. 5, pp. 666–677, 2012.

[4] Manvi, S. S. and Krishna Shyam, G. "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey,**" *J.*** *Netw. Comput. Appl*., vol. 41, no. 1, pp. 424–440, 2014.

[5] Hussin,M. Abdullah,A. and Subramaniam, S. "Adaptive Resource Allocation for Reliable Performance in Heterogeneous Distributed Systems," *Algorithms Archit. Parallel Process*., pp. 51–58, 2013.

[6] Dhinesh Babu, L. D. and Venkata Krishna, P. "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Appl. Soft Computer Journal*, vol. 13, no. 5, pp. 2292–2303, 2013.

[7] Ryan, T. and Choon Lee, Y. "Multi-Tier Resource Allocation for Data-Intensive Computing," *(2015) Big Data Res*., vol. 1, pp. 1–7, 2015.

[8] Hung, P. P. and Huh, E. "An Adaptive Procedure for Task Scheduling Optimization in Mobile Cloud Computing," *Math. Probl. Eng*. (2015).

[9] Moschakis, I. a. and Karatza, H. D. "A meta-heuristic optimization approach to the scheduling of bag-of-tasks applications on heterogeneous clouds with multi-level arrivals and critical jobs," *Simul. Model. Pract. Theory*, vol. 57, pp. 1–25, 2015.

[10] Babu, L. D. D., Gunasekaran, A. and Krishna, P. V. "A decision-based pre-emptive fair scheduling strategy to process cloud computing work-flows for sustainable enterprise management," *Int. J. Bus. Inf. Syst*., vol. 16, no. 4, pp. 409, 2014.

[11] Hung, P. P. , Van Nguyen, M., Aazam, M. , and Huh, E. "Task Scheduling for Optimizing Recovery Time in Cloud Computing*," in Computing, Management and Telecommunications (ComManTel)*, 2014 International Conference on, pp. 188-193, 2014.

[12] Javadi, B. , Abawajy, J. , and Buyya, R. "Failure-aware resource provisioning for hybrid Cloud infrastructure," *J. Parallel Distrib. Comput.*, vol. 72, no. 10, pp. 1318–1331, 2012.

[13] Javadi, B. , Thulasiraman, P. and Buyya, R. "Cloud Resource Provisioning to Extend the Capacity of Local Resources in the Presence of Failures," *2012 IEEE 14th Int. Conf. High Perform. Comput. Commun*. 2012 IEEE 9th Int. Conf. Embed. Softw. Syst., pp. 311–319, 2012.

[14] Hussin, M. and Latip, R. "Adaptive Resource Control Mechanism Through Reputation-Based Scheduling in Heterogeneous Distributed Systems," *J. Comput. Sci*., vol. 9, no. 12, pp. 1661–1668, 2013.

[15] Islam, M. , Balaji, P., Sadayappan, P. , and Panda, D. "QoPS: A QoS based scheme for parallel job scheduling," *Job Sched. Strateg. Parallel Process.*, pp. 252–268, 2003.

[16] Yousaf, M. M. and Welzl, M. "Network-aware HEFT scheduling for grid.," *ScientificWorldJournal*., pp. 317284, 2014.

[17] Li, J. , Qiu, M. , and Niu, J. "Adaptive resource allocation for preemptable jobs in cloud systems," *Intell. Syst. Des*., pp. 31–36, 2010.