Indian Journal of Science and Technology, *Vol 8(12), DOI: 10.17485/ijst/2015/v8i12/57984, June 2015*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Scheduling Workflow Applications on the Heterogeneous Cloud Resources

R. Bagheri[1] and M. Jahanshahi[2*]

[1]Department of Computer Engineering, Gazvin Branch, Islamic Azad University, Tehran, Iran
[2]Department of Computer Engineering, Central Tehran Branch, Islamic Azad University, Tehran, Iran; mjahanshahi@iauctb.ac.ir

Abstract

As cloud computing model recently become promising and enables users to obtain their required services, many users desirous to run their workflow applications on it. Scheduling workflow is one of the most important challenges in the cloud. For optimal use of the capabilities of the distributed system, an efficient scheduling algorithm is needed. Addressing the problem of scheduling workflow applications onto Cloud environment is the main contribution of this paper. Heterogeneity of resource types is one of the most important issues which significantly affect workflow scheduling in Cloud environment. On the other hand, a workflow application is usually consisting of different tasks with the need for different resource types to complete which we call it heterogeneity in workflow. The main idea in this paper is to match the heterogeneity in workflow application to the heterogeneity in Cloud environment. To obtain this objective a new scheduling algorithm is introduced, which is based upon the idea of detecting the set of tasks that could run concurrently and distribute them into different sub-workflows and then allocate each sub-workflow in resource cluster instead of allocating individual tasks. This can reduce inter-task communication cost and thus improve workflow execution performance. First we perform global scheduling and then conduct local scheduling. On the Global-scheduling to achieve high parallelism the received DAG partition into multiple sub-workflows that is realized by WPRC algorithm. On the Local-scheduling, sub-workflows were generated at the global level are dispatched to selected resource clusters. We used the simulation to evaluate the performance of the proposed algorithm in comparison with three well-known approaches. The results show that the proposed algorithm outperforms other approaches in different QoS related terms.

Keywords: Cloud Computing, Cluster resources, Scheduling Algorithm, Workflow

1. Introduction

Cloud computing is the service-focused that delivers hardware infrastructure and software application as services with low cost and high quality[1,2]. These technology advances have led to the possibility of using geographically distributed to solve large-scale problems in science, engineering, and commerce. Currently, cloud computing services are categorized into three classes: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). These services are available in a pay-per-use on demand model[3,4]. Some researchers consider the benefits of using cloud computing for executing scientific workflows[5–8]. Several features that are distinct cloud computing from other computing environments consist of: (1) The type and number of compute resources assigned to a workflow are determined by service requests. (2) Compute resources in Cloud are exposed as services that provide a standardized interface for services to access over the network[9]. Workflows constitute a common model for describing a wide range of scientific applications in distributed systems. Usually, a workflow can be represented by a Directed Acyclic Graph (DAG) in which each computational task is represented

*Author for correspondence

by a node, and each data or control dependency between tasks is represented by a directed edge between the corresponding nodes. Workflow scheduling is the way of choosing a suitable resource for each task. With Growing up and complexity of workflow, the need of workflow's scheduling be felt more than ever and has become one of the most important challenges in the cloud. The workflow scheduler has to schedule and allocate each task according to the dependency of the workflow's tasks.

As task scheduling is a well-known NP-complete problem[10]. Many heuristic methods have been proposed for distributed system. Most of them try to minimize the total completion time of all tasks (make span) and cost of the workflow[11–14]. Zeng et al. proposed a budget-conscious scheduler to minimize many-task workflow execution time within a certain budget[15]. In[16], Abrishami et al. designed a QoS-based workflow scheduling algorithm based on Partial Critical Paths (PCP) in SaaS clouds to minimize the cost of workflow execution within a user defined deadline. Workflow scheduling algorithms are classified into four classes: 1) list-based, 2) duplication-based, 3) clustering-based, and 4) level-based. Most previous workflows scheduling research are based on a list-based heuristic approach. List-based scheduling is a class of scheduling heuristics in which tasks are assigned with priorities and placed in a list ordered in decreasing magnitude of priority. An important issue in DAG scheduling is how to rank the nodes. The rank of a node is used as its priority in the scheduling and then allocates each individual task onto processors. Many list-based heuristics proposed in the literature[14,17,18]. In this paper, we propose an algorithm according to the clustering-based heuristic approach. Clustering is another efficient way to reduce a communication delay in DAGs by grouping heavily communicating tasks to the same labeled clusters[19]. In general, clustering algorithms have two phases: the task clustering phase that partitions the original task graph into clusters for allocation instead of allocating individual tasks. This can reduce inter-task communication cost and thus improve workflow execution performance and post-clustering phases which can refine the clusters produced in the previous phase and get the final task-to-resource map. The rest of the paper is organized as follow: Section 2, describes the scheduling architecture and model used by algorithm. Section 3, the proposed scheduling algorithms are explained. Section 4, evaluates simulation results. Section 5, concludes.

2. Workflow Scheduling Model

The proposed workflow scheduling model consists of a Cloud application model and a cloud resource model. A Cloud application is modeled by a Directed Acyclic Graph (DAG), G = (V, E, q, w), in which V = $\{t_i \mid i = 1, 2, \ldots, m\}$ be the finite set of tasks t_i and E be the set of directed arcs of the form $e(t_i, t_j)$, An edge $e(t_i,t_j)$ represents the communication from t_i to t_j where t_i is called a prior of t_j , and t_j is a successor of t_i and q_i represents the computational cost of task t_i. The weight w of $e(t_i,t_j)$ represents the communication cost from task t_i to t_j.

In cloud resource model, resources have been clustered that formed with multiple VMs. Resource clusters are connected by a WAN, and within a cluster, computational nodes are connected by high speed LAN. Usually, the communication cost by WAN is much higher than LAN. We name the cost of communication between two resources in different cluster as external communication cost similarly; we name the cost of communication within a cluster, internal communication cost. The internal data transfer in cloud similar to the most recently publish is negligible. Each resource cluster represented by C_i ($1 \le i \le m$), where i is the unique identification number of the cluster and m is the number of clusters.

When a workflow receives, it will break into subgraphs, according to knowledge about available resources, by executing the Workflow Partition Resource Clusters (WPRC) algorithm and then scheduler will run the local level scheduling and map tasks in subgraph to local computational node. A computational resource is denoted as $R_{i,j}$ where i is the resource cluster id to which this resource belongs and j is the resource id within its cluster. For a resource cluster C_i, the number of resources in C_i is represented by n_i. Let $p_{i,j}$ is the expected performance of $R_{i,j}$ so P_i is total computation power of C_i, which is the sum of the computational power of all resources in C_i:

$$Pi = \sum_{j=1}^{ni} P_{i,j} \tag{1}$$

The average performance of all available Computational resources in C_i is given by:

$$\overline{P_i} = \frac{P_i}{n_i} \tag{2}$$

$\overline{P}$ is the average computational power of all resource clusters that compute by:

$$\overline{P} = \frac{1}{m} \sum_{1<i<m} \overline{P_i} \tag{3}$$

We assumed that communication cost between resource cluster C_i and C_j is represented by *Com Cost*_$C_{i,j}$ and the average cross-cluster communication cost is defined as:

$$Com\ Cost_{C_i} = \frac{1}{m^2} \sum_{i,j=1}^{nl} Com\ Cost_{C_{i,j}} \tag{4}$$

We assumed that the computation power of resource effect on the time required completing a task and the time to finish a data transfer is commensurate with the communication cost of the link.

2.1 WPRC: Scheduling Algorithm

High parallelism means to dispatch more tasks simultaneously to different resources. To achieve this, the main task graph needs to be partitioned into subgraphs and each subgraph has to be assigned to a resource cluster**.** The main parameter must be determined in partitioning of graph, is the number of partitions should be made (N). To determine N, CTC parameter is used. *CTC* is the ratio of communication-to-computation. A high *CTC* value means a task graph is computation intensive. Formally, *CTC* is defined as:

$$CTC = \frac{\overline{P}}{\overline{q}} \tag{5}$$

Where $\overline{q}$ is the average processing requirement of all tasks. As the *CTC* increases, high parallelism is preferred because more computational power is required. WPRC determine the number of graph partitions to be created, *N*, according to different workflow patterns and communication costs:

$$N = \min\left(m, \left\lceil CTC / \left(ComCost_{C_i}\right) \times \beta \right\rceil\right) \tag{6}$$

$$\beta = \frac{\Delta p}{\overline{P}} \tag{7}$$

$$\Delta p = \sqrt{\frac{1}{m} \sum_{1<i<m} \left(\overline{P} - \overline{P_i}\right)^2} \tag{8}$$

Here β is the accelerating factor and Δp is the standard deviation on the computational power of different resource clusters. It is clear that *N* is always no greater than the number of available resource clusters.

With the number of subgraphs, to be created, WPRC is to specify how tasks in the main graph should be assigned. To achieve high parallelism and avoid inessential external communication, the size of a subgraph assigned to a resource cluster should be as large as possible under a certain threshold value. The weights of edges connecting different subgraphs should be as small as possible to minimize communication cost. According to purpose, we need to detect the set of tasks that could run concurrently and distribute them into different subgraphs and then specify the maximum number of nodes that could run concurrently when assigned to the same resource cluster that call *Maximum Concurrent Node* (*MCN*). To get *MCN*, two parameters are defined: For a node t_i in a DAG, its *Earliest Start Time (EST)* is defined as follows:

$$\begin{aligned} &EST\ (t_i) = \max\ (ET\ (t_j) + e\ (t_j, t_i)) \\ &t_j \in Pred\ (t_i) \end{aligned} \tag{9}$$

Where *pred* (t_i) is the set of immediate predecessors of t_i and Execution Time (ET) of t_j is defined as:

$$ET\left(t_j\right) = \frac{q_j}{\overline{P}} \tag{10}$$

The Earliest Finish Time (EFT) of t_i is defined as follows:

$$EFT\ (t_i) = EST\ (t_i) + ET\ (t_i) \tag{11}$$

Now, we can specify which nodes could run concurrently. We call t_i and t_j parallel peers, if following equation is satisfied to one of them: if the *EST*(t_i) after *EST* (t_j) and before the EFT (t_j) or the EFT (t_i) after *EST* (t_j)and before the EFT (t_j). By checking parallel peers of every node, we can find the largest set of concurrent nodes in task graph *G*, whose size is the value of *MCN*. The size of a partition is also related with the computational power of resource clusters. We assume that the set of resource cluster services that are offered include:

$$S = \{S_1, S_2 \ldots S_m\} \tag{12}$$

If the number of resources in cluster *i* that offer service *j* is $n_{i,sj}$,the average number of all available resources in cluster i for service j is given by:

$$N = \frac{n_{i,sj}}{n_i} \tag{13}$$

Sum of Computing power of all resources in the cluster *i* that offer service *j* is Computation power (p_{sj}), the average Computation power of all available resources in cluster i for service j is given by:

$$P_{sj} = \frac{P_{sj}}{\sum_{i=1}^{m} P_{si}} \tag{14}$$

Resource failures to be statistically independent and follow a constant *failure rate* Fr_j for each resource j. We consider the reliability of an activity i as the Probability of successful completion on a resource j, modeled using an exponential distribution[20,21]:

$$R\left(T_i, R_j\right) = e^{-Fr_j * exetime(T_i, R_j)} \tag{15}$$

Total reliability of all resources in the cluster *i* that offer service *j* is given by:

$$R_i\left(S_j\right) = \sum_{\substack{Rj \in \text{resource} \\ \text{deliver service j}}} R(T, R_j) \tag{16}$$

Where, *T* is the average of tasks computation. The average reliability of all available resource resources in cluster i for different service is given by:

$$R\left(S_j\right) = \frac{R_i\left(S_j\right)}{\sum_{j=1}^{m} R_i\left(S_j\right)} \tag{17}$$

Then for each resource cluster, a weight (W) can be obtained for each type of service in the resource cluster by the following equation:

$$W_{ij} = N + \text{Computation power}\left(P_{sj}\right) + R\left(S_j\right) \tag{18}$$

After we get the weight of all services in the all resource clusters, we then create the clusters matrix:

$$\begin{array}{c} \\ cls1 \\ cls2 \\ . \\ . \\ . \\ clsn \end{array} \begin{array}{c} \begin{array}{cccc} s1 & s2 & ... & sm \end{array} \\ \begin{bmatrix} W11 & W12 & ... & W1m \\ W21 & W22 & ... & W2m \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ Wn1 & Wn2 & ... & Wnm \end{bmatrix} \end{array}$$

In which W_{ij} is weight of the resource cluster *i* for the service *j*.

To be adaptive to the dynamic and heterogeneous nature of the computational cloud, WPRC introduces two parameters to describe the related properties of a resource cluster, namely the *Cluster Rank* (R_i) and *Parallel Threshold* (T_i). Thus

$$R_i = 1 - \frac{1}{2W_i} \sum_{j=1}^{n_i} \left(d_{i,j} + n_i \Delta S_i\right) \tag{19}$$

Where, $d_{i,j}$ is the standard deviation of the performance fluctuation of $P_{i,j}$ in vari ous time slots, and W_i is sum of weights of all resources in cluster *i* for various services that can be obtain by the following equation:

$$W_i = \sum_{j=1}^{m} W_{i,j} \tag{20}$$

And ΔS_i is the standard deviation of the computational capacity of resources in r_i (so a larger $d_{i,j}$ means the performance of $p_{i,j}$ is more unstable and a larger ΔS_i implies that r_i is more heterogeneous). After resource clusters are ranked, *N* out of *m* of them, having the *N* highest ranks is selected for the current job. We assume these clusters are r_1, r_N. Then the initial threshold value T_i' of r_i is defined as:

$$t_i = n_i \times R_i \times \frac{\sum_{j=1}^{m} \frac{W_{i,j}}{m}}{\sum_{i=1}^{N} \left(\sum_{j=1}^{m} \frac{\left(\frac{W_{i,j}}{m}\right)}{N} \right)} \tag{21}$$

The above equation, the parallel computation power of each cluster is also taken into account. Then the parallel threshold value T_i of each subgraph to be created is given by:

$$T_i = \frac{t_i}{\sum_{i=1}^{N} t_i} MCN(G) \tag{22}$$

The pseudo-code for the WPRC is given in Algorithm1. WPRC is described as follows. When a scheduler receives a job, it first traverses the job's DAG *G* to compute its *CTC*, the number of partitions to create *N*, and the level of each task node. Then, the scheduler selects *N* resource clusters whose ranks are the highest *N* out of *m*, according to its knowledge. A graph partition iteration checks every remaining nodes in *G* to determine whether the node can be put into a subgraph *G'*.

Algorithm 1 WPRC

Input: A task DAG *G* (*V, E*) and available Cloud resource clusters $C_1 \ldots C_m$.
Output: A subgraph of *G* and assigned to resource cluster C_i.

1. Compute *CTC*, *N* and *EFT* and *EST* of each node in *G*, and cluster ranks *R* and threshold values *T*;
2. Mark all nodes unassigned;

3. Select the resource cluster C_i with the highest threshold value *Ti*;
4. Find the largest edge *e* (*a*, *b*) in which *a*, *b* have not been checked;
5. *IF Br* (*G'* + *a*) ≤ *Ti* and *Br* (*G'* + *b*) ≤ *Ti* {
6. Add *a and b* to *G' and* mark *a and b* as checked;
7. }
8. *G* = *G* - *G'*;
9. Put (*G'*, C_i) in the output set.

WPRC itself does not give the task-to-resource map, but on the global level only partitions original workflow to different subgraphs and then dispatches subgraphs to different resource clusters. So, on the local level, another scheduling algorithm is needed to get the final schedule of the received subgraphs. For scheduling a subgraph in special resource cluster need to rank all nodes and then assign node to resource according to its priority. Usually, the priority of a task node can be obtained by finding the maximum distance from this node to the starting node. Distance means the sum of computational and communication costs along a certain path. To estimate the completion time of nodes, we use the average performance value of resource cluster. The rank or priority of a node is defined as:

$$\text{rank}(t_i) = \max_{t_j \in \text{Pred}(t_i)} \left\{ \text{rank}(t_j) + \left(\frac{q_j}{\bar{p}} \right) + e(i,j) * \text{inter}_{\text{cost}} \right\} \quad (23)$$

For the tasks to resources mapping, initially priority of each task is computed. Then at each step, a task that has the highest priority in the ready queue RQ is selected. The task is placed in a ready queue RQ if all its predecessors have been implemented and the middle results are provided. Once the task node is selected, the function *Select Processor* (ti) for choice suitable source is called. Mapping algorithm adopts a forward-looking approach to decision-making based on only the current state of resources and task. Function Select Processor (t_i) compute the execution time of task t_i for all the resources that provide the required service t_i. It can be obtained from the following equation:

$$ET(t_i) = q_i / \bar{P} \quad (24)$$

After the task execution time were computed from all sources, then the resource, where is minimize, is selected. After each assignment, the rank of tasks will be update. The pseudo-code for the Mapping is given in Algorithm 2.

Algorithm 2 Mapping

Input: A task graph *G* and a set of resources $R_1 \ldots R_n$
Output: A mapping of tasks to resources

1. Compute *rank* for each task;
2. Initialize the ready queue *RQ* with the entry task;
3. WHILE (there are unscheduled nodes) {
4. Select the highest priority task t_i in *RQ*
5. Call *Select Processor* (t_i) to assign task t_i
6. Update priorities of all tasks
7. }

Select Processor (task t_i)

1. FOR all available resource R_j that deliver service type x (ti request service type x) {
2. Compute ET(t_i)
3. }
4. Select the resource that has min ET (t_i);
5. Insert t_i to selected Resource;
6. Update available resource;

3. Discussion

In this section, we will present our simulation of the Workflow Partition Resource Clusters algorithm.

3.1 Simulation Model

We evaluate the performance of WPRC using CloudSim[22], which has been widely adopted for the modeling and evaluation of cloud-based solutions. In the experiments, three resource clusters are used. Each cluster consists of different resources number connected by a LAN. The resource clusters are connected by a WAN. In each resource cluster, the resources in it have different computing power and delivered different type of services. In terms of input task graphs, We used random graph generation with the ability of generating a variety of task graphs according to different configuration parameters, such as average number of task nodes of each graph, average outgoing and incoming degrees for each node in a graph, and computational and communication cost for each type of task nodes and edges. The count of nodes in each workflow graph set between 25 and 100 nodes. Each graph has a single entry and a single exit node. We used the random

graph generator discussed in [17]. This random graph generator requires following input parameters:

- *V*: The number of task in the DAG.
- Out degree: The ratio of maximum out edges of a node to total nodes of the DAG.
- Communication to Computation Ratio (CCR). It is the ratio of the average communication cost to the average computation cost.
- *B*: The computational heterogeneity factor of resources.
- α: The depth parameter of the DAG. This parameter indicates the depth of a DAG by using the uniform distribution with the mean value equal to $\frac{\sqrt{V}}{\alpha}$.

The values for the input parameters are shown in Table 1. The following four metrics are used to evaluate the performance of the proposed algorithm:

3.2 Simulation Results

To evaluate the performance of the proposed algorithm we compared it with the HEFT[17], the QRS[23] and the PFAS[24] which are well-known methods for DAG scheduling. The performance metric used in the simulation is the average make span. To test our proposed algorithm, the following parameters are considered in the experiment: 1) The average number of task nodes in a graph *v* 2) The ratio of the average degree of a task node to the total number of tasks in a graph (Edge density in a graph) 3) *CCR*.

Figure 1 present the average make span of our approach over the others approach with respect to different number of task.

Figure 2 present the average makespan of our approach over the others approach with respect to different degree.

Figure 3 present the average make span of our approach over the others approach with respect to different CCR.

The result of simulation indicates that the average make span of workflow's graph with our approach algorithm is better than other algorithms.

Table 1. The values for the input parameters

Parameter	Value
V	20, 40, 60, 80, 100
Out degree	0.1, 0.2, 0.3, 0.4, 0.5
CCR	0.1, 0.5, 1.0, 5.0, 10.0
α	0.5, 1.0, 2.0
B	0.1, 0.25, 0.5, 0.75, 1.0

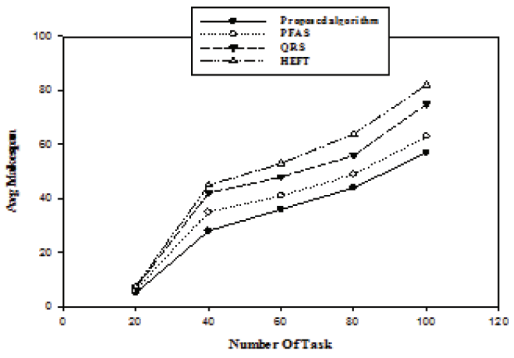

Figure 1. Avg Makespane with different number of task.

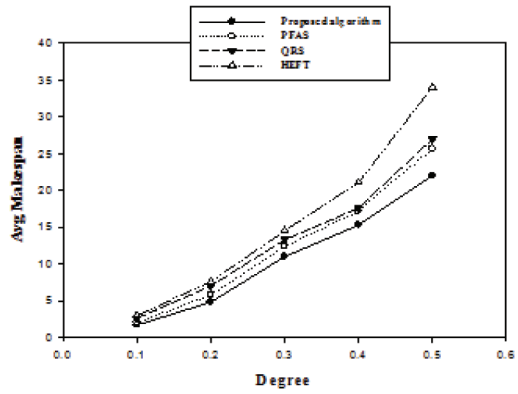

Figure 2. Avg Makespane with different degree.

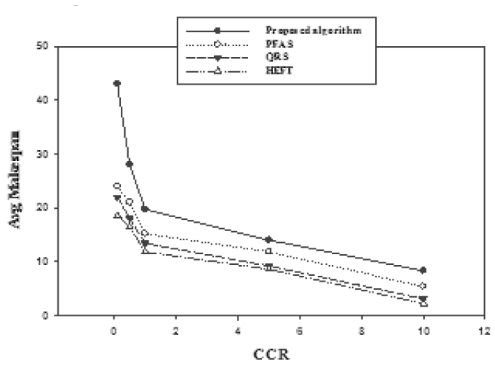

Figure 3. Avg Makespane with different CCR.

3.3 Further Analysis

Because our proposed algorithm is based on clustering method and it is base on concurrent tasks, we define tasks Graph Concurrency degree (GC[a]) that can be obtained from the following equation:

$$GC = \frac{MCN}{NT} \quad (25)$$

[a]Graph Concurrency

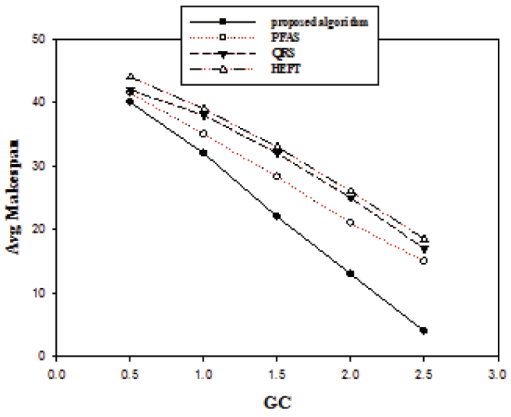

Figure 4. Avg Makespane of workflows with different GC and increasing Difference the Avg Makespane with increases GC.

Where MCN is the maximum number of tasks in a workflow's graph that can be executed in parallel and NT is the Number of tasks in a workflow. Simulation results in Figure 4 show that whatever amount GC becomes larger (ratio the number of concurrent tasks to the total tasks) Difference in performance of the proposed algorithm with other algorithms will be more and improvement of the proposed algorithm compared to other algorithms will be more.

4. Conclusion

We propose a new Workflow Partition algorithm WPRC for a workflow scheduling in the cloud. Scheduling is in two phases, on the global level, WPRC clusters workflow to achieve high parallelism and on the local level, Sub workflows was generated at the global level are then dispatched to selected resource clusters. It not only considers the heterogeneity and dynamism of cloud resources, but also uses an adaptive strategy according to different workflow patterns and resource topologies. When it partitions a task graph, WPRC try to minimizing the cost of workflow execution and the make span. Future work includes improving the current strategy, and uses other parameters for clustering the workflow.

5. References

1. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems. 2009; 25(6):599–616.
2. Rajathi A, Saravanan N. A survey on secure storage in cloud computing. Indian Journal of Science and Technology. 2013; 6(4):4396–401.
3. Verma A, Kaushal S. Cloud computing security issues and challenges: a survey. Advances in Computing and Communications. 2011; 193:445–54.
4. Pal AS, Pattnaik BPK. Classification of virtualization environment for cloud computing. Indian Journal of Science and Technology. 2013; 6(1):3965–71.
5. Deelman E. Grids and clouds: making workflow applications work in heterogeneous distributed environments. International Journal of High Performance Computing Applications. 2010; 24(3):284–98.
6. Hoffa C, Mehta G, Freeman T, Deelman E, Keahey K, Berriman B, et al. On the use of cloud computing for scientific workflows. eScience, 2008 eScience'08 IEEE Fourth International Conference; IEEE; 2008.
7. Juve G, Deelman E, Vahi K, Mehta G, Berriman B, Berman BP, et al. Scientific workflow applications on Amazon EC2. 2009 5th IEEE International Conference on E-Science Workshops; IEEE; 2009.
8. Sathick KJ, Jaya A. Natural Language to SQL Generation for Semantic Knowledge Extraction in Social Web Sources. Indian Journal of Science and Technology. 2015; 8(1):1–10.
9. Lin C, Lu S. Scheduling scientific workflows elastically for cloud computing. 2011 IEEE International Conference on Cloud Computing (CLOUD); IEEE; 2011; p. 746–7.
10. Yu J, Buyya R, Ramamohanarao K. Workflow scheduling algorithms for grid computing. Metaheuristics for scheduling in distributed computing environments. 2008; 146:173–214.
11. Gu Y, Wu Q. Optimizing distributed computing workflows in heterogeneous network environments. Distributed Computing and Networking. 2010; 5935:142–54.
12. Rahman M, Venugopal S, Buyya R. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. IEEE International Conference on e-Science and Grid Computing; IEEE; 2007. P. 35–42.
13. Wu Q, Gu Y. Optimizing end-to-end performance of data-intensive computing pipelines in heterogeneous network environments. Journal of Parallel and Distributed Computing. 2011; 71(2):254–65.
14. Sakellariou R, Zhao H. A hybrid heuristic for DAG scheduling on heterogeneous systems. 2004 Proceedings 18th International Parallel and Distributed Processing Symposium; IEEE. 2004.
15. Zeng L, Veeravalli B, Li X. Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud. 2012 IEEE 26th International Conference on Advanced Information Networking and Applications (AINA); IEEE; 2012.

16. Abrishami S, Naghibzadeh M. Deadline-constrained workflow scheduling in software as a service cloud. Scientia Iranica. 2012; 19(3):680–9.
17. Topcuoglu H, Hariri S, Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems. 2002; 13(3):260–74.
18. Kwok Y-K, Ahmad I. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. IEEE Transactions on Parallel and Distributed Systems. 1996; 7(5):506–21.
19. Yang T, Gerasoulis A. DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Transactions on Parallel and Distributed Systems. 1994; 5(9):951–67.
20. Dogan A, Ozguner F. Trading off execution time for reliability in scheduling precedence-constrained tasks in heterogeneous computing. Proceedings 15th International Parallel and Distributed Processing Symposium; IEEE; 2001.
21. Yu J, Buyya R, Tham CK. Cost-based scheduling of scientific workflow applications on utility grids. e-Science and Grid Computing, 2005 First International Conference; IEEE. 2005.
22. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience. 2011; 41(1):23–50.
23. Chunlin L, Xiu ZJ, Layuan L. Resource scheduling with conflicting objectives in grid environments. J Netw Comput. 2009; 3:760–9.
24. Dong F, Akl SG. PFAS: a resource-performance-fluctuation-aware workflow scheduling algorithm for Grid Computing. 2007 IPDPS 2007 IEEE International Parallel and Distributed Processing Symposium; IEEE. 2007.