Vol. 5(17) Oct. 2015, PP. 2485-2499

Contents list available at IJMEC

International Journal of Mechatronics, Electrical and Computer Technology (IJMEC)

Journal Homepage: www.aeuso.org

PISSN: 2411-6173 - EISSN: 2305-0543

Scientific Workflow Scheduling Based on Deadline Constraints in Cloud Environment

Elahe Donyadari[1*], Faramarz Safi Esfahani[2] and Nasim Nourafza[3]

[1, 2, 3] *Department of Computer Engineering, Najafabad Branch, Islamic Azad University, Najafabad, Iran*

***Corresponding Author's E-mail:** e.donyadari@gmail.com*

Abstract

Cloud computing is providing an environment for scientific workflows where large-scale and complex scientific analysis can be scheduled onto a heterogeneous collection of computational and storage resources. A scientific workflow is described as a paradigm, which is used to describe a set of structured activities and scientific computations. Scientific workflow scheduling has become one of the most challenging issues in cloud systems. Scheduling of scientific workflow applications involves the mapping of tasks to computational resources, based on quality of service requirements such as time, cost, bandwidth, etc. Most of the proposed scheduling algorithms require detailed information about tasks, e.g., execution time, and remaining time. On the other hand, the most proposed algorithms cannot schedule tasks in the shortest possible time by using minimum knowledge about tasks. In this article, we introduce an approach for task scheduling, namely RRRSD (Relation aware Round Robin Scheduling based on Deadline constraints). It applies the Round Robin algorithm along with deadline parameters. The main goal of this model is to optimize the mapping of tasks to available resources in order to minimize makespan time and the failure rate of scientific workflows. The simulation results show an average improvement of 24.25% for makespan time of workflows and the failure rate of 36.21% compared to four basic scheduling algorithms.

Keywords: *Cloud Computing, Scheduling, Scientific Workflow, Deadline.*

1. Introduction

Cloud computing is a convenient model based on shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) [1]. It is a parallel-distributed system that includes a large number of interconnected virtual computers. Scientific workflows are often used in large-scale complex e-science applications such as climate modelling, biology, chemistry, and astronomy simulation [2]. Compared with business workflows, scientific workflows involve massive computation and data analysis. Distributed systems such as clouds and grids will be helpful for scientific workflows processing by allocating optimum resources and decreasing costs of computation and storage [3]. Task Scheduling is a major problem in scientific workflows execution. The scheduling algorithms are presented to balance the load on CPUs, maximize CPU usage, and to minimize a workflow total execution time. The main goal of task scheduling is finding proper tasks mapping to available resources [2], [3]. The

Article History:

Received Date: Apr. 25, 2015
Accepted Date: Jul. 17, 2015
Available Online: Sep. 19, 2015

computation time of existing methods is high because all the information about tasks such as execution time and remaining time must be known before the scheduling and partitioning process.

The current research proposes a scheduling model based on deadline constraints. The main goals of this model are reducing execution time and failure rate of scientific workflows in cloud environments under constant budget. The novelty of our research lies in completion of maximum number of tasks before missing their deadline time, using the minimum information about the tasks. Business workflows are not considered in this model. We hypothesized that using the Round Robin algorithm and task partitioning with deadline constraints will reduce the execution time of scientific workflows.

The main contributions of this paper are summarized as follows:

- Partitioning tasks according to the parent-child relationship in a task graph and assigning them as a group to the virtual machines.
- Using the Round Robin algorithm for task execution regardless of task size.
- Scheduling tasks considering their deadlines in order to execute the maximum number of tasks before missing their deadlines
- Minimizing failure rate as well as the execution time of workflows.

The rest of the article is organized as follows:
In section 2, the related research works are reviewed briefly. In section 3, the proposed model is described. The simulation results are presented in section 4, and finally conclusion and future works are summarized in section 5.

2. Related Works

In this section, we summarize related research works, which schedule tasks based on deadline constraint. A task scheduling model based on the Round Robin algorithm is proposed by [4]. This model uses two separated queues for scheduling process: a queue for the tasks waiting for execution and another for those running at least on one machine. This model sets the priority to the bag-of-tasks in the execution queue based on execution cost and then runs them based on the Round Robin algorithm. Bag-of-tasks is a virtual bag, which has at least two tasks and each bag has a specific execution time for every task it involves. There is a drawback in using the bag-of-tasks. If bags of tasks are executed successfully, the execution time of the model is reduced, but if bags of tasks fail to be executed, the execution time and number of failed tasks are increased.

SC-PCP (Software as a Service - Cloud Partial Critical Path) [5] and IC-PCP (Infrastructure as a Service - Cloud Partial Critical Path) [6] are two workflow scheduling algorithms based on quality of service (QoS) factors. These algorithms concentrate on reducing workflow execution costs based on the deadline defined by the user, using the Partial Critical Paths algorithm. In order to reduce the total workflow execution time, tasks are allocated to the resources that process them in the shortest possible time. Like these algorithms, we use partitioning method, but our method is completely different from theirs.

A PSO (Particle Swarm Optimization) based framework is designed by [7] for workflow scheduling in the cloud environment to obtain an optimal solution for task scheduling. This framework uses factors like workflow deadline, virtual sample number, virtual sample type, and necessary operation type for workflow scheduling. It provides solutions for purchasing virtual machines and operations with an optimum budget for any specified scientific workflow application.

A model for efficient management of workflow ensembles under budget and deadline constraints on Infrastructure as a Service (IaaS) are suggested by [8],and [9]. These models use one static algorithm, which schedule the workflows before combining them, and two dynamic algorithms. If the workflow has a higher capacity (cost and deadline time) than the specified capacity of system, the model will not allow

the dynamic algorithms to schedule workflows. Lack of attention to data storage capacity and data transfer costs are major drawbacks of these model.

SLPSO (Self-adaptive Learning PSO algorithm) uses the integer-programming model for efficient resource management during the scheduling process of bag-of-tasks [10]. This model tries to satisfy QoS factors (especially tasks' deadlines) by updating new patterns of particle global best and personal best in PSO. Although the implementation cost of SLPSO is high, this model shows high performance in scheduling large-sized tasks.

An architecture named "RMS" (Resource Management System) is proposed by [11], to schedule applications and bag-of-tasks dynamically, in the hybrid cloud. RMS is based on the Max-Min algorithm and considers the factors such as deadline, resource allocation monitoring, system workload, and task execution time. RMS can increase the total utilization of public cloud services without any impact on the capacity of meeting application deadlines [11]. Reducing the execution time is not the objective of this framework.

DBD-CTO (Deadline and Budget Distribution based Cost-Time Optimization) schedules workflows using deadlines and budgets constraint which are defined by the user [12]. In DBD-CTO, workflow paths are divided into several partitions according to the user-defined processing time, and cost. Similar to this model, our model objective is to execute maximum number of tasks before missing their deadlines under user-defined budget.

A dynamic real time scheduler is designed by [13]. This scheduler can optimize resource utilization by task monitoring during the scheduling process. It calculates time slices for optimal task scheduling on virtual machines based on execution time, deadline time, and statistical concepts. It has a high rate of hardware resource utilization in cloud and grid systems. Nevertheless, the low performance in dynamic virtualization models is a drawback of this model.

PDSA (Prioritized Deadline-Based Scheduling Algorithm) schedules tasks based on the delay time [14]. The delay time is the difference between burst time and deadline time. PDSA executes tasks in round robin manner after assigning priority to them. It has an efficient performance in variety of synthetic workloads. However, PDSA has no consideration for reduction of execution cost.

Three different scheduling models named LSTRR (Least Slack time Round Robin Based Scheduling Algorithm), SPTFRR (Shortest Processing Time First Round Robin Based Scheduling Algorithm), and EDFRR (Earliest Deadline First Round Robin Based Scheduling Algorithm) are presented by [15] for scheduling tasks in Grid environments. These models schedule the tasks according to minimum execution time, slack time, earliest deadline time, minimum completion time, and task arrival time. Then execute them in round robin fashion. These algorithms provide an efficient approach for scheduling tasks when the number of processors is increased, but long waiting time for task execution is the major drawback of these algorithms.

Two real time job scheduling algorithms based on map/reduce model are presented by [16],and [17]. In both algorithms, a central scheduler estimates probability of job execution on virtual machines according to their deadline time, and then shares them between map/reduce tasks. PDCS (Preemptive Deadline Constraint Scheduler) [17] outperforms the proposed algorithm in [16], when the scheduling process involves jobs which have computational intensive map/reduce tasks. In general, PDCS has lower completion time and better time slots utilization than the other proposed algorithm.

An online scheduling model in cloud environment, which tries to optimize resource utilization by combining the V-Dover algorithm and deadline time is proposed by [18]. In addition, the authors proposed the transformation of job parameters and the capacity functions that reduce the offline

varying capacity problem to the constant capacity problem [18]. Model efficiency is evaluated in online systems under the synthetic workload. The results show the high performance in overloaded systems. Unlike this algorithm, our model is an offline scheduling algorithm.

The research work in [19] evaluates LJFS (Largest Job First Served) and AFCFS (Adaptive First Come First Serve) algorithms. These algorithms perform group scheduling. AFCFS algorithm uses the FCFS algorithm features, but the difference between AFCFS and FCFS is that AFCFS considers idle or busy modes of virtual machines. LJFS organizes tasks according to their size and then executes them. Both algorithms demonstrate that they could be applied efficiently in an environment with a dynamic number of VMs (Virtual Machines) [19]. Although, both algorithms provide similar performance for medium workloads, LJFS outperforms AFCFS when workloads get heavier [19]. LJFS provides a superior cost-performance efficiency than AFCFS in situations of heavy workload.

A robust scheduling algorithm with resource allocation policies that schedule workflows on heterogeneous Cloud resources while trying to minimize the total elapsed time and the cost is presented by [20]. This scheduling algorithm maps tasks on heterogeneous Cloud resources and judiciously adds slack time based on the deadline and budget constraints to make the schedule robust [20]. Additionally, three multi-objective resource selection policies are presented, which maximize robustness while minimizing execution time and cost [20]. These policies select the best solution from the feasible solution for each PCP (Partial Critical Path). The results show that the proposed resource allocation policies provide robust and fault-tolerant schedule while minimizing execution time [20].

3. Proposed Model

3.1 Main Idea

The classic Round Robin algorithm emphasizes on execution of tasks with minimum information about them. This algorithm uses a circular queue to store tasks and a fixed time quantum for task execution. If a task does not complete during its time quantum, it returns to the waiting queue and waits for its next time quantum. The algorithm benefit is that the tasks are executed by turn and do not wait for completion of other tasks.

The model presented in this paper guarantees the completion of maximum number of tasks before missing their deadline time using the minimum information about them to minimize the execution time and failure rate of the scientific workflows.

3.2 Basic Definitions

In the RRRSD model, a workflow is described by a Directed Acyclic Graph (DAG). This graph consists of homogeneous task groups (jobs) that are related by similarity of functions. Each workflow is symbolized by W = (T, E), where T is a set of n tasks (nodes) and E is a set of data dependencies (edges). We assume that the budget of workflow execution is a constant value. Each workflow has an overall deadline, and each task in the workflow has its own deadline, which are symbolized by ODL, and DL, respectively. The user who submits the workflow in the system, defines each task deadline (DL) and an overall deadline (ODL). All tasks must be finished before missing ODL time. Turnaround time is computed according to Equation (1):

$$\textit{Turnaround Time (i)} = \textit{FT (i)} - \textit{ST (i)} \qquad 1 \leq i \leq n \tag{1}$$

Where ST is the start time, and FT is the finish time of task execution. In the RRRSD model, execution of each task can be started anytime, but it must be finished before missing its own deadline and the overall deadline.

The Average Makespan Time is defined as the average turnaround time of all tasks in a workflow, and makespan is calculated by Equation (2):

$$Average\ Makespan = \frac{\sum_{i=1}^{n}(Turnaround\ Time(i))}{n} \qquad 1 \leq i \leq n \qquad (2)$$

Where n is the number of tasks.
A task is possible to be successfully executed; while, its turnaround time does not violate its DL and the ODL. The Remaining Time to Violation of Deadline (RTDV) for each finished task is calculated by Equation (3) to determine whether the execution status of a task is successful or failed:

$$RDTV\ (i) = DL\ (i) - Turnaround\ Time\ (i) \qquad 1 \leq i \leq n \qquad (3)$$

The numbers of failed tasks that are missing their deadline are defined as Failed Jobs (FJ) and the number of tasks meeting their deadline are defined as Success Jobs (SJ). These parameters are calculated by Equation. (4):

$$\begin{cases} if\ RTVD(i) \geq 0\ then\ SJ = SJ + 1 \\ if\ RTVD(i) < 0\ then\ FJ = SJ + 1 \end{cases} \qquad 1 \leq i \leq n \qquad (4)$$

It is possible that a waiting task in the execution queue is likely to be close to miss its deadline. In order to refrain from this situation, Deadline Ratio parameter (DR) is considered. This parameter is calculated in each time quantum. If the DR of a waiting task is lower than the current running task, it means that the waiting task is close to miss its deadline. In this case, the running task is suspended, and the task with a lower DR is executed, subsequently. The DR for each task is computed by Equation (5):

$$Deadline\ Ratio\ (i) = \frac{DL(i)/\ VM.MIPS + Current\ Time}{ODL} \qquad 1 \leq i \leq n \qquad (5)$$

Where MIPS (Millions of Instructions Per Second) is the number of instructions performed by a virtual machine, and the current time is current system time.

Success Rate (SR) and Failure Rate (FR) are calculated by Equation (6), and Equation (7), respectively:

$$Success\ Rate = \frac{SJ}{n} \times 100\% \qquad (6)$$

$$Failure\ Rate = \frac{FJ}{n} \times 100\% \qquad (7)$$

Improvement is calculated by Equation (8):

$$Improvement = \frac{Average\ Makespan\ Time_{\ RRRSD} - Average\ Makespan\ Time_{\ Other\ Method}}{Average\ Makespan\ Time_{\ RRRSD}} \qquad (8)$$

Another parameter is Degree Dependency (DD), which is defined as the sum of input, and output edges connected to each task. In this way, a task with the highest degree dependency has higher priority than the others for scheduling tasks on available resources. The position of each task in a Directed Acyclic Graph is denoted as a Level. We use the horizontal clustering, which merges the tasks in the same level into the jobs to reduce the scheduling overheads. In addition, we consider a separate queue for each job in every level. Figure 1(a) shows the original workflow, and Figure 1(b) shows the workflow after applying horizontal clustering.

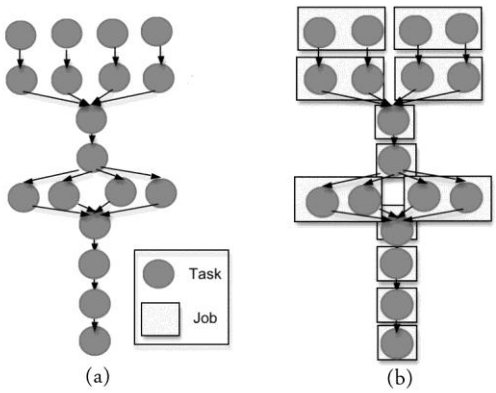

Figure 1: (a) Original workflow, (b) the workflow after horizontal clustering.[21]

3.3 Task Placement Rules

There are three task placement rules in the RRRSD scheduling model as follows:

Rule 1. Child tasks must be executed on the same virtual machine as their direct parent is scheduled on it, as much as possible. A child task can be executed on another idle virtual machine if its parent's virtual machine is busy.

Rule 2. If two tasks at the same level have equal deadlines, then the task that has a higher data dependency will be scheduled first.

Rule 3. The deadline ratio of tasks in each queue must be updated in every time quantum.

3.4 Task Partitioning

In order to improve task assignment, we propose a static task-partitioning algorithm. This algorithm partitions the tasks based on parent-child relationship. First, the algorithm recognizes all parents and children by using the Parallel Depth First Search algorithm [22]. Then it partitions the direct parents and their children into the smaller groups recursively. At first step, the parent and its directly connected children at the lowest level are placed in one group. Afterward, this group is associated with the next level parent and makes a bigger group. In this way, a compound group is created at each step. This process is continued until all nodes are grouped. Figure 2(a) shows the workflow before partitioning and Figure 2(b) shows the workflow after applying our task-partitioning algorithm. In addition, a field is created for each task, which shows its direct parent after applying our partitioning algorithm. During the scheduling process, this field is used to assign the tasks to virtual machines.

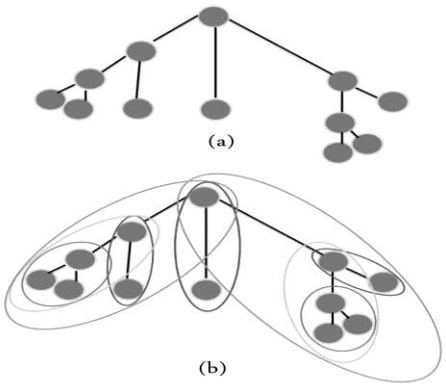

Figure 2: (a) Workflow before task partitioning, (b) workflow after task partitioning.

3.5 Task Scheduling

The RRRSD model is divided into three steps as follows:

Step1. First, direct parents and their children in workflow are grouped via partitioning algorithm. Then, the dependency degrees (DD) are calculated for each task based on the workflow type.

Step2. Horizontal clustering is applied over the workflow and tasks are merged into the jobs. After that, all tasks in a job are classified in a queue and sorted based on their deadline. Then the tasks are compared with each other according to their deadline (DL) and dependency degree (DD), to identify the order of task execution. The tasks with the highest data dependency and lowest deadline time have higher priority than the other tasks.

Step3. The tasks belonging to each queue are assigned to a virtual machine based on their priority considering Rule 1 and Rule 2, which are described in section 3.3.

It is notable that the jobs in the same level have no data dependency and they are processed in parallel, but each level is dependent to its upper and lower levels and they should be executed consequently. We suggest an approach to parallel the execution of consequent levels. If required data of tasks in related levels is prepared before the completion of current level execution, then current level will pass the prepared data to related levels and they could start their execution in parallel with current level. In this way, the number of executed tasks will be increased and the execution time would be reduced.

According to the experiments, which are done in Pegasus [23], Kepler [24], and Trident [25], the average time of changing status of each task in scientific workflows during the execution process is estimated between 10-50 milliseconds. Hence, we consider the time quantum as 10 milliseconds.

At the end of each time quantum, a deadline ratio (DR) is calculated for the current running task and all other tasks based on Equation (5). If the deadline ratio (DR) of a waiting task is lower than the current running task, then the running task will be suspended, and the execution of waiting task will be started; otherwise, the current task does not suspend and its execution will be continued. The tasks are executed according to the scheduling parameters, which are described in section 3.2 in round robin manner. The scheduling process will be continued until all tasks in different levels are executed. During the scheduling process, the status of virtual machines is continuously checked. If the status of a virtual machine is changed to idle, the virtual machine reports its status to the data center. Then the data center assigns the idle virtual machine to a waiting task that is located in waiting queues. This strategy increases the utilization rate of virtual machines.

According to Equation (4), if the task misses its deadline, the system will be in a state of failure. Two fault tolerance approaches for handling this failure are considered:

1. The task, which misses its deadline is inserted into the end of its queue and would have another chance to be executed before missing the overall deadline. If the task misses the overall deadline, it would be considered as a system failure.
2. The task, which misses its deadline, is deleted from the execution queue. Then the user submits the task to the system again and defines a new deadline for it.

The RRRSD scheduling model applies the first approach. Second approach will be considered as a future work. The advantage of the RRRSD scheduling model is in respect to the parent-child relationship or data dependencies, which are very important in processing of scientific workflows. In addition, calculating the deadline ratio (DR) in each period causes the maximum number of tasks to be executed before missing the overall deadline; therefore, the success rate of this model will be increased. Figure 3 shows a

flowchart of the RRRSD scheduling model and Figure 4 shows the components of the RRRSD scheduling model and the interactions between them.

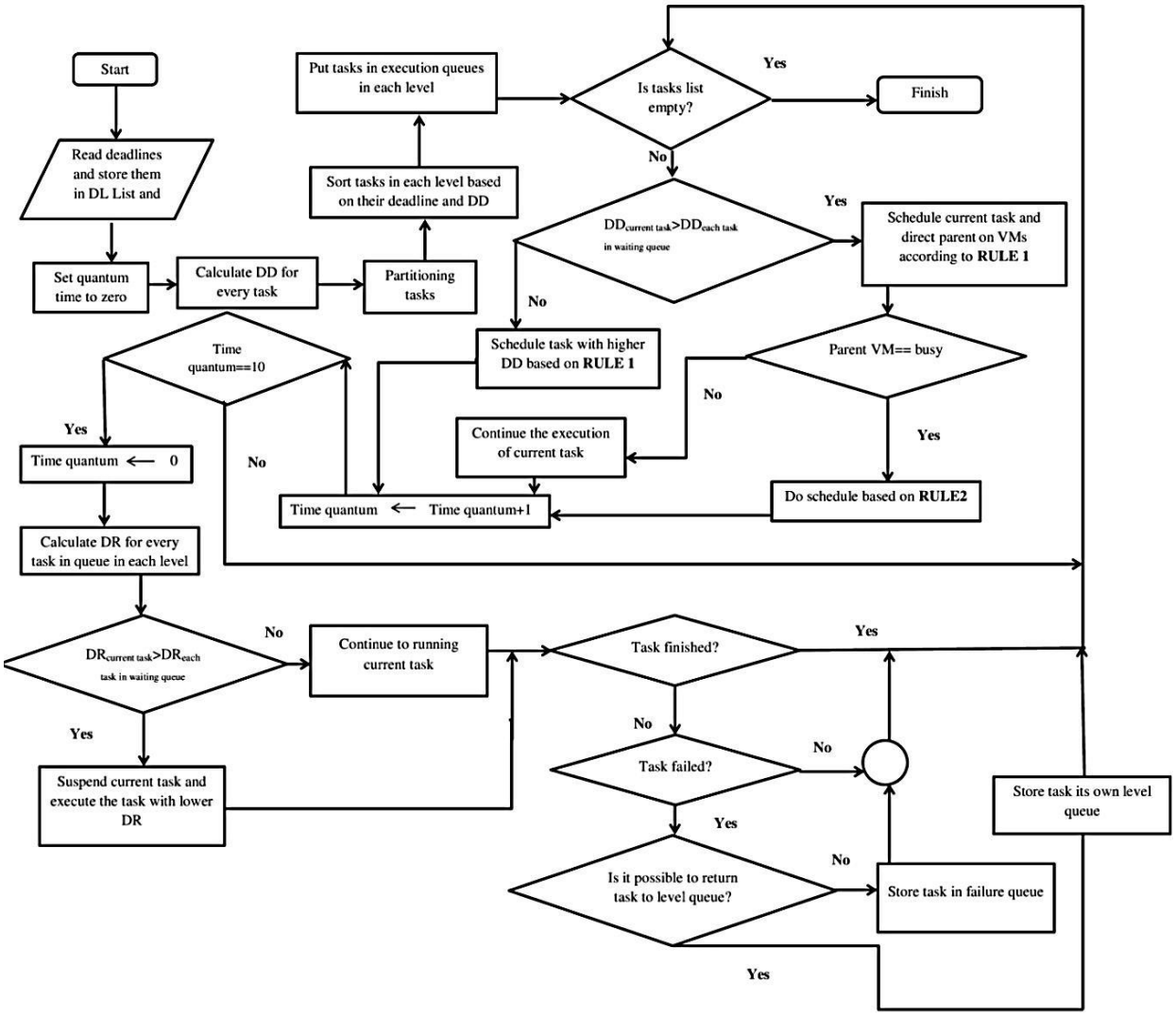

Figure 3: Flowchart of the RRRSD model

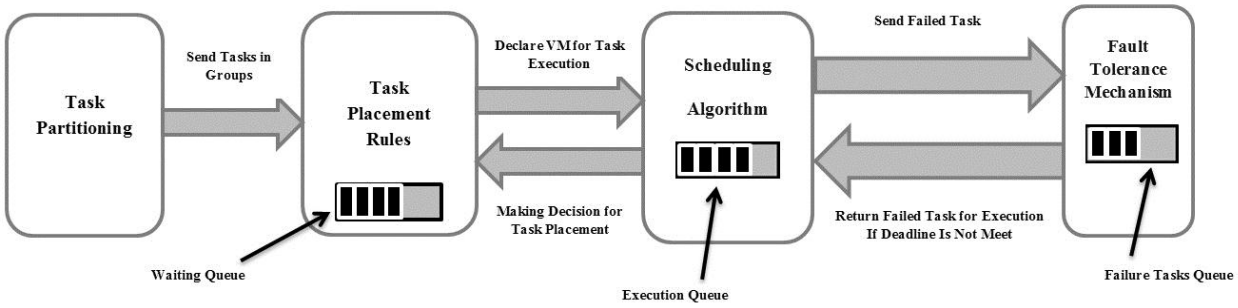

Figure 4: The RRRSD scheduling model components

4. Evaluations

In this section, we present the simulation results of the RRRSD scheduling model.

4.1 Experimental Environment

Workflowsim [21] is a Java-based workflow engine that has been developed based on Cloudsim core [26]. It has been extended CloudSim by providing a workflow level support of simulation. It has been used for scientific workflow management in a cloud environment. In addition, in this research, we have used it for the same goal. The Workflowsim simulator consists of data centers, which involve virtual machines to execute workflows. The execution time of each task type on its virtual machine is set to task turnaround time. In addition, the execution cost for each workflow is a constant budget. Figure 5 shows all the components of the RRRSD model.

To evaluate our workflow scheduling model, we have used a library of realistic workflows introduced by Bharathi et al. [27]. They studied the structure of five realistic workflows from diverse scientific applications, which are: Montage (astronomy), Cybershake (earthquake science), Epigenomics, LIGO (gravitational physics),and SIPHT (biology) [5]. Figure.6 shows the small instance of each workflow structure. For each workflow, the tasks with the same color are of the same type and can be processed by a common service or virtual machine.

Furthermore, Bharathi et al. have developed a workflow generator, which can create synthetic workflows of arbitrary size, similar to real world scientific workflows [5]. Using this workflow generator, they can create different sizes for each workflow application in terms of total number of tasks. These workflows are available on their website (https://confluence.pegasus.isi.edu/display/pegasus/Workflow Generator). We have chosen two sizes for our experiments from mentioned website. They are Large (about 1000 jobs), and Extra Large (about 6000 jobs) as special cases for SIPHT and Epigenomics. Experimental parameters are shown in Table 1.

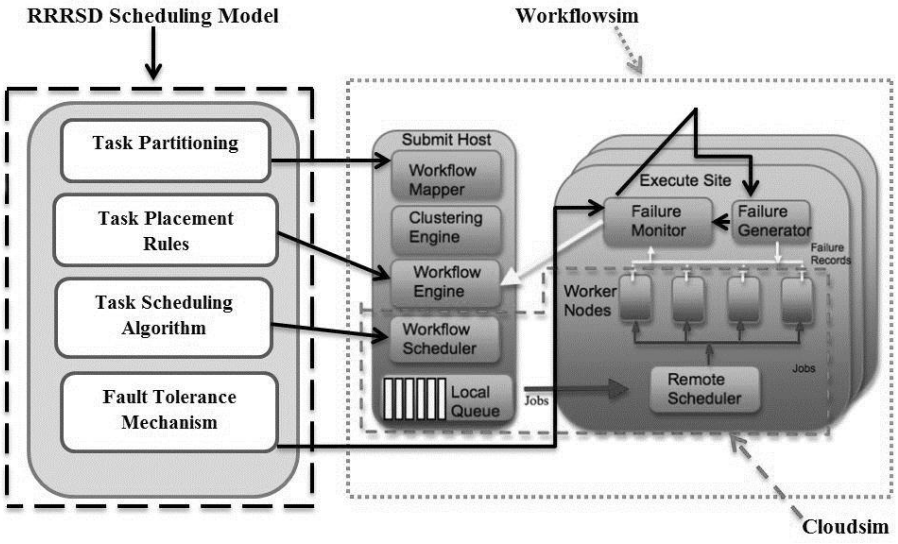

Figure 5: Components of the RRRSD scheduling model

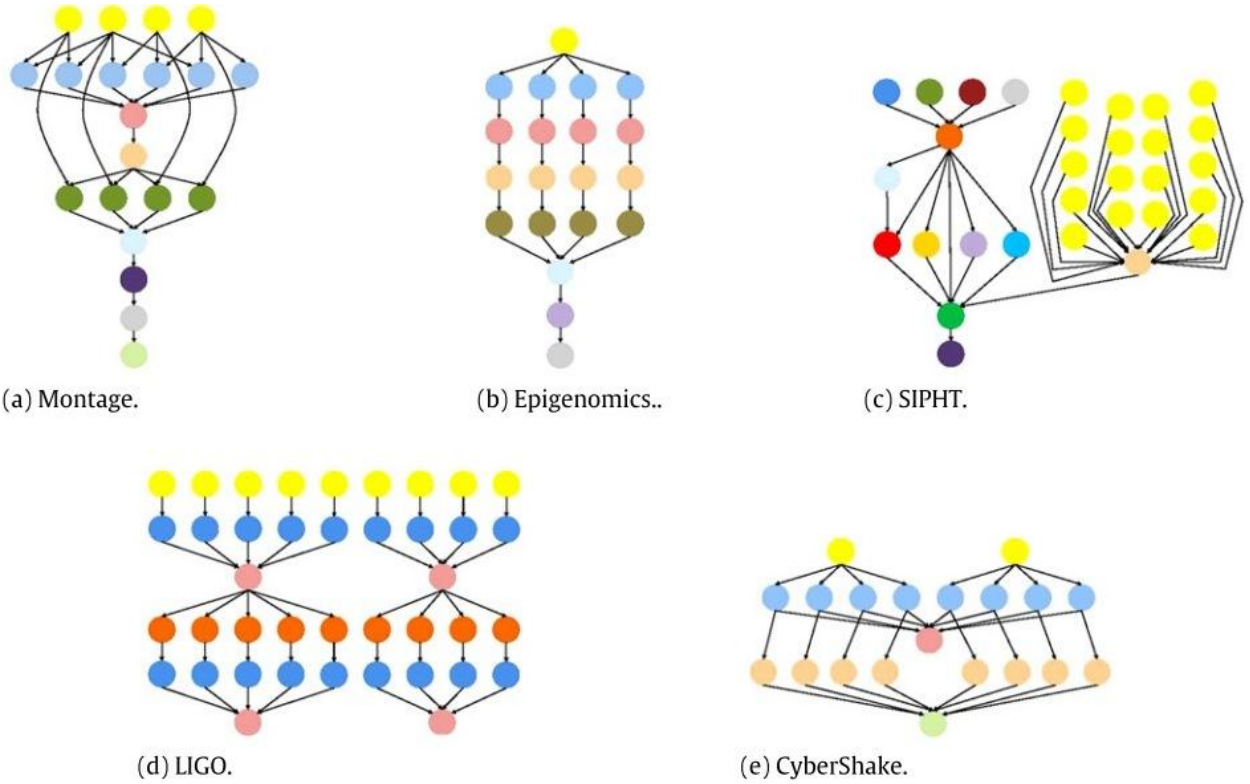

Figure 6: The structure of five realistic scientific workflows [5], [27].

Table 1: Different configuration for experimental environment

Number of Data Center	**2**		
Number of Virtual Machines	**25**		
Bandwidth	**1024**		
MIPS	**1024**		
Deadline	**5000 millisecond**	**1250 millisecond**	**625 millisecond**
Number of Tasks	**1000**	**6000**	
Budget	**9240$**		

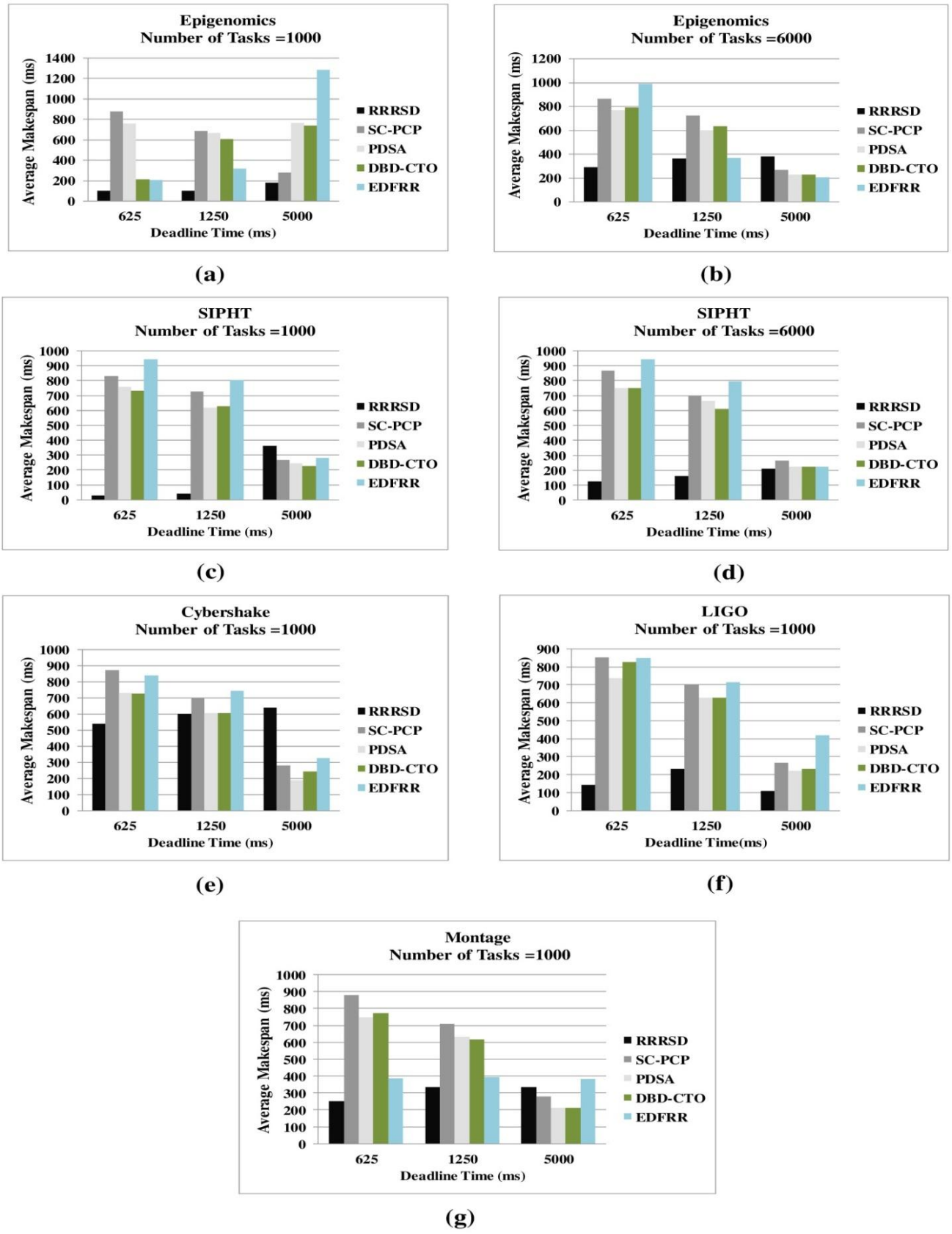

Figure 7: (a-g) Experimental results in different configuration.

4.2 Experimental Results

We implemented SC-PCP [5], DBD-CTO [12], PDSA [14], and EDFRR [15] algorithms. Afterward, we tested these scheduling algorithms and our RRRSD scheduling model on the data sets that were introduced in section 4.1. Finally, we compared their results with our scheduling model. Figure7 shows the comparison results in different configurations.

Workflows were executed on the condition that the deadline time was changed. The RRRSD scheduling model scheduled maximum number of tasks successfully before missing their deadline with the average failure rate of 3.167%. However, SC-PCP, DBD-CTO, PDSA, and EDFRR scheduled workflows with the average failure rates of 18.42%, 43.85%, 49.14%, and 33.71% respectively.

Epigenomics: As shown in Figure 6, Epigenomics workflow consists of paths with equal length, but the tasks in these paths have different deadline time. All of the mentioned algorithms finished Epigenomics scheduling process before the overall deadline times except PDSA, DBD-CTO, and EDFRR when the number of tasks was 1000, and the deadline time was 625ms. Although SC-PCP, DBD-CTO, PDSA, and EDFRR had better makespan times than our scheduling model, when the number of tasks was 6000 and deadline time was 625ms, but these algorithms did not execute some tasks before missing their deadlines and their failure rate is more than our model. Using the Longest Path algorithm in SC-PCP, paths have no superiority over each other and large tasks might be preferred to small tasks; therefore, it had a higher makespan time than our model. PDSA and EDFRR have long waiting time for the large tasks; therefore, they had higher makespan time than our model. Finally, putting tasks into the bags with no attention to the priority of the tasks is the reason of high makespan time in DBD-CTO. Our model had an average failure rate of 9.5% and maximum number of tasks was executed successfully. Due to its tasks priority based on deadline ratio, the maximum number of tasks was executed before missing their deadline. In addition, while task numbers were 1000, the makespan time of RRRSD was notably less than the described algorithms. Figure 7(a) and Figure 7(b) show the results of the Epigenomics workflow under different deadline time.

SIPHT: based on Figure 6, this type of workflow has two separated parts that their intermediate results are combined in the last task to produce the final result. DBD-CTO, EDFRR, and PDSA assign tasks to any available resource without considering the processing time of virtual machines. In general, DBD-CTO, EDFRR, and PDSA have no strategy for efficient resource management. In SC-PCP, partitioning and scheduling tasks based on the Longest Path algorithm give more priority to one part of workflow than the other part and thus parallel and fair execution of SIPHT workflow are not considered in this algorithm. Our model schedules the tasks with respect to the parent-child relationships in a DAG; hence, it has proper task distribution among idle virtual machines. In addition, calculating the deadline ratio (DR) gives a priority to each task; therefore, two parts of SIPHT workflow are scheduled fairly. As shown in Figure7 (c), and Figure 7(d), the RRRSD makespan time was significantly improved for all workflow sizes in comparison with the four evaluated algorithms.

Cybershake: The results indicate that when the deadline time was 625ms, the makespan time of RRRSD was higher than the other described algorithms. Nevertheless, when the deadline time was increased, the RRRSD's makespan time was improved comparing with others. In DBD-CTO, and PDSA there is no particular order for task placement on virtual machines, hence the failure rate was increased during the experiments. In SC-PCP, large tasks are preferred to small tasks; therefore, small tasks have a long waiting time. Hence, it had a higher makespan time compared with the RRRSD scheduling model. Based on Figure 6, there are many small tasks with approximately equal deadline time in many levels of Cybershake workflow (such as level2, and level4). Using horizontal clustering, we combined these tasks into a job; therefore, in our model small tasks and large tasks are scheduled based on their deadline ratio (DR) fairly. Figure 7(e) shows the average makespan time of Cybershake workflow.

LIGO: LIGO workflow consists of two or more blocks that each block generates specific results on a subset of data. The results must be generated at the earliest possible time. DBD-CTO, PDSA, and EDFRR have no strategy for task distribution on virtual machines, so their makespan time is higher than our model. SC-PCP applies the FCFS (First Come First Serve) algorithm to execute LIGO's blocks e.g., at first it executes the first block, then the second one and so on. It uses the Longest Path algorithm for partitioning tasks based on the execution cost of paths in each block. In general, due to using the FCFS algorithm, there is no strategy for parallel execution of blocks in SC-PCP and tasks in follow up blocks have long waiting time.

The RRRSD is capable to schedule tasks, without considering the blocks of task, using deadline ratio (DR) factor; hence, it has a better task distribution on VMs in comparison to the other evaluated algorithms. In other words, RRRSD puts all the tasks placed on the same level of blocks in one execution queue. After that, by using horizontal clustering, it reduces the mentioned tasks to two or more jobs. Comparing the RRRSD scheduling model with the other evaluated algorithms, RRRSD had a higher makespan time when deadline time was 5000ms. However, when the deadline time was decreased, makespan time of the RRRSD was improved compared with other described algorithms. Figure 7(f) shows the average makespan time of LIGO workflow under different deadline time.

Montage: Like Cybershake workflow, Montage workflow consists of many levels that have small tasks with approximately equal deadline time (such as level1, and level2). In RRRSD scheduling model task priority was calculated based on deadline time and data dependency, so small tasks had no superiority over large tasks. The PDSA and EDFRR had a long waiting time for the large tasks; therefore, they had a higher makespan time than the RRRSD model. Partitioning the tasks based on the Longest Path algorithm and sharing them between the bags, without any attention to the tasks priority during the scheduling process are the reasons of high makespan time in SC-PCP and DBD-CTO, respectively. Based on Figure 7(g), there was very little difference in the makespan time between RRRSD and other evaluated algorithms for this workflow when the deadline time was 625ms. However, RRRSD showed significant improvement in makespan time than other algorithms when the deadline time was increased.

In general, RRRSD had more efficient resource management method than the other described algorithms. The simulation results showed that RRRSD had a better degree of parallelism in Epigenomics and SIPHT workflows compared to the other evaluated algorithms. Table 2 presents failure rate in percent for RRRSD scheduling model and the other evaluated algorithms when deadline time varies 5000ms, 1250ms, and 625ms. Table 3 presents average improvement of RRRSD scheduling model compared to the other evaluated algorithms.

Table 2: Failure rates (%) of the four evaluated algorithms compared with RRRSD scheduling model

	Montage	Epigenomics		Cybershake	LIGO	SIPHT	
Number of Tasks	**1000**	**1000**	**6000**	**1000**	**1000**	**1000**	**6000**
RRRSD	0%	1%	18%	0.17%	1%	1%	1%
SC-PCP	0%	2%	28%	75%	1%	7%	16%
PDSA	38%	75%	58%	65%	25%	59%	24%
DBD-CTO	40%	41%	49%	65%	65%	25%	22%
EDFRR	13%	42%	36%	55%	39%	36%	15%

Table 3: Average improvement of RRRSD scheduling model compared to the four evaluated algorithms.

Type Of Workflow	Cybershake	Montage	LIGO	SIPHT		Epigenomics	
Number of Tasks	1000	1000	1000	1000	6000	1000	6000
PDSA	0.168	0.658	0.637	7.420	2.145	5.045	0.480
SC-PCP	0.016	0.949	0.888	8.610	2.507	3.738	0.719
DBD-CTO	0.135	0.659	0.696	7.270	2.036	3.785	0.528
EDFRR	0.052	0.284	1.261	9.461	2.711	4.611	0.434

Conclusion and Future Works

This article addresses the problem of scientific workflow scheduling in the cloud environment. We proposed a RRRSD scheduling model for scheduling scientific workflows based on deadline constraints using the Round Robin algorithm and workflow partitioning. Experimental results showed that the average makespan time and failure rate of RRRSD scheduling model were significantly improved compared to the four selected algorithms. Since the scientific workflows are generally composed of many tasks, the RRRSD scheduling model is considerably efficient for executing these workflows before missing their deadline. We suggest an interactive fault tolerance mechanism that interacts with the users while failure occurs. In addition, considerations of task size as well as deadline constraint are suggested for future research works.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Future Generation computer systems, vol. 25, pp. 599-616, 2009.

[2] A. Barker and J. Van Hemert, "Scientific workflow: a survey and research directions," Parallel Processing and Applied Mathematics, Springer, 2008, pp. 746-753.

[3] L. Singh and S. Singh, "A Survey of Workflow Scheduling Algorithms and Research Issues," International Journal of Computer Applications, vol. 74, pp. 21-28, 2013.

[4] S. Bansal, B. Kothari, and C. Hota, "Dynamic Task-Scheduling in Grid Computing using Prioritized Round Robin Algorithm," International Journal of Computer Science, vol. 8, pp. 472-477, 2011.

[5] S. Abrishami and M. Naghibzadeh, "Deadline-constrained workflow scheduling in software as a service Cloud," Scientia Iranica, vol. 19, pp. 680-689, 2012.

[6] S. Abrishami, M. Naghibzadeh, and D. Epema, "Deadline-constrained workflow scheduling algorithms for IaaS Clouds," Future Generation Computer Systems, vol. 29, pp. 158-169, 2012.

[7] N. Netjinda, T. Achalakul, and B. Sirinaovakul, "Cloud Provisioning for Workflow Application with Deadline using Discrete PSO," ECTI TRANSACTIONS ON COMPUTER AND INFORMATION TECHNOLOGY vol. 7, pp. 44-52, 2013.

[8] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, p. 22.

[9] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, "Cost optimization of execution of multi-level deadline-constrained scientific workflows on clouds," Parallel Processing and Applied Mathematics, Springer, 2014, pp. 251-260.

[10] X. Zuo, G. Zhang, and W. Tan, "Self-Adaptive Learning PSO-Based Deadline Constrained Task Scheduling for Hybrid IaaS Cloud,"IEEE Transactions onAutomation Science and Engineering, vol. 11, pp. 564 - 573 2013.

[11] R. N. Calheiros and R. Buyya, "Cost-Effective provisioning and scheduling of deadline-constrained applications in hybrid clouds," Web Information Systems Engineering-WISE 2012, Springer, 2012, pp. 171-184.

[12] A. Verma and S. Kaushal, "Deadline and Budget Distribution based Cost-Time Optimization Workflow Scheduling Algorithm for Cloud," International Conference on Recent Advances and Future Trends in Information Technology, 2012.

[13] O. Khalid, I. Maljevic, R. Anthony, M. Petridis, K. Parrott, and M. Schulz, "Deadline aware virtual machine scheduler for grid and cloud computing," IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2010, pp. 85-90.

[14] H. A. Abba, S. N. M. Shah, N. B. Zakaria, and A. J. Pal, "Deadline Based Performance Evaluation of Job Scheduling Algorithms," International Conference in Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC) , pp. 106-110.
[15] H. A. Abba, N. B. Zakaria, A. J. Pal, and K. Naono, "Performance Comparison of Some Hybrid Deadline Based Scheduling Algorithms for Computational Grid," Advances in Information Technology, Springer, 2012, pp. 19-30.
[16] K. Kc and K. Anyanwu, "Scheduling hadoop jobs to meet deadlines," IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010, pp. 388-392.
[17] L. Liu, Y. Zhou, M. Liu, G. Xu, X. Chen, D. Fan, W.Qianru, "Preemptive Hadoop Jobs Scheduling under a Deadline," Eighth International Conference on Semantics, Knowledge and Grids (SKG), pp. 72-79.
[18] S. Chen, T. He, H. Y. S. Wong, K.-W. Lee, and L. Tong, "Secondary job scheduling in the cloud with deadlines," IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011, pp. 1009-1016.
[19] I. A. Moschakis and H. D. Karatza, "Evaluation of gang scheduling performance and cost in a cloud computing system," The Journal of Supercomputing, vol. 59, pp. 975-992, 2012.
[20] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao, "Robust scheduling of scientific workflows with deadline and budget constraints in clouds," IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), 2014, pp. 858-865.
[21] W. Chen and E. Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," IEEE 8th International Conference on E-Science (e-Science), 2012, pp. 1-8.
[22] V. N. Rao and V. Kumar, "Parallel depth first search. Part I. implementation," International Journal of Parallel Programming, vol. 16, pp. 479-499, 1987.
[23] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, M.Gaurang, V.Karan, B.Berriman,and J.Good "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Scientific Programming, vol. 13, pp. 219-237, 2005.
[24] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, L.Edwards, T.Jing, and ZH.Yang, "Scientific workflow management and the Kepler system," Concurrency and Computation: Practice and Experience, vol. 18, pp. 1039-1065, 2006.
[25] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, and Y. Simmhan, "The trident scientific workflow workbench," IEEE Fourth International Conference on eScience, 2008, pp. 317-318.
[26] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, pp. 23-50, 2011.
[27] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.H. Su, and K. Vahi, "Characterization of scientific workflows," Third Workshop in Workflows in Support of Large-Scale Science, 2008, pp. 1-10.