ISSN No. 0976-5697

Volume 8, No. 5, May – June 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Reliable Task Scheduling Based on Resource Availability in Cloud Computing

Chitra S
Research Scholar
Dept. of Computer Science and Engineering
New Horizon College of Engineering
Bangalore, India
chitraravi.cr@gmail.com

Dr. Prashanth C S R
Professor and Head
Dept. of Computer Science and Engineering
New Horizon College of Engineering
Bangalore, India
drprashanthcsr@gmail.com

Abstract— With ever increasing scale of cloud computing systems, to execute compute intensive parallel applications, in an environment where cloud resource availability is dynamic, effective task scheduling with reliability is an important factor to be considered. Virtual machines provided by current cloud infrastructures do not exhibit a stable performance due to network connectivity as well as computational nodes and may have a significant impact while scheduling workflows on clouds. Since the resources are not dedicated and can be used by other users simultaneously, there are load variations on resources, resulting in fluctuations in resource availability that affects the schedules. In this paper, a new reliable task scheduling algorithm based on resource availability is proposed, where task scheduling decision is based not only on task ready time but the processor available time slots that affect finish time of task. The main objective of the proposed technique is to reduce the overall execution time without increasing resource need. When algorithms that assume full resource availability are executed in that scenario, resource non-available slots contribute to delays thus increasing makespan. The proposed algorithm takes into consideration the resource availability factor during the prediction interval of application execution, while mapping tasks to resources during scheduling decisions, thus increasing reliability.

Keywords: Directed Acyclic Graphs, Makespan, Reliability, Resource Availability, Schedule Length Ratio, Speedup, Time slots

I. INTRODUCTION

In heterogeneous distributed computing systems like grids and clouds, resources are shared heavily. Also virtualization and heterogeneity of non-virtualized hardware in clouds result in a variability in the performance of resources[1] [2]. Quite complex scientific and business workflow applications consisting of multiple tasks with precedence constraints, are executed in such environments. In such systems, effective task scheduling is a very important concern for the execution of performance driven applications. Virtual machines in clouds use underlying hardware such as processor, memory, power, storage and network bandwidth, etc. They are in different autonomous domains and the computing power available for large applications varies over time. Each virtual machine is allocated a portion of the underlying hardware resources by the hosts, based on factors such as user defined resource limits, total available resources for the host, the number of virtual machines powered on and resource usage by those virtual machines, overheads for virtualization[3]. In dynamic environments, task execution time estimates become unpredictable[4]. Information regarding the availability levels of resources must be considered while scheduling the tasks. Schedulers that effectively predict the availability of resources, and use these predictions to make scheduling decisions, can improve performance. Ignoring resource availability characteristics can lead to delays in makespan and can affect reliability[5]. A probabilistic estimate of resource availability can be considered while making scheduling decisions.

With scale of cloud computing systems increasing, reliability is an important factor to be considered in order to execute compute intensive parallel applications. Many static list scheduling algorithms like HEFT[6], PETS[7], ECTS[8] in literature only consider minimizing makespan assuming resources are always available based on earliest finish time. The performance variation of Virtual Machines (VM) in Clouds affects the overall execution time (i.e. makespan) of the workflow[9] due to variations in CPU performance and network resources[1]. It also increases the difficulty to estimate the task execution time accurately[10].This indeterminism makes it extremely difficult for schedulers to estimate runtimes and make accurate scheduling decisions to fulfil user Quality of Service (QoS) requirements. Other sources of uncertainty in cloud platforms that contribute to unreliable task execution are VM provisioning and deprovisioning delays[1]. Resource monitoring can be done to collect the resources information like CPU load, memory usage, network bandwidth[10]. This information could be used for task scheduling, which in turn will improve performance of scheduling algorithm.

We propose a new Reliable task Scheduling algorithm based on resource availability (RSRA) in cloud computing systems. Resource availability is predicted with resource monitoring tools for a certain prediction interval depending on the probability of availability of the underlying hardware to the Virtual machines. Available and non-available time slots are determined, and this is considered in the VM selection phase while making scheduling decisions, making it more reliable. The main objective of the proposed method is to reduce the overall execution time without increasing resource need. The actual task execution times that take into account the resource availability factor increases reliability in getting parallel applications executed in such systems. This algorithm is compared with existing algorithms which

© 2015-19, IJARCS All Rights Reserved

assume full resource availability while making scheduling decisions and is found to be more reliable and performing better than the existing algorithms when executed in the scenario considering resource availability factor.

Section 2 specifies problem statement. Section 3 gives an overview of application model, resource model and workflow scheduling algorithms as background. Section 4 contains related work. In Section 5 the proposed RSPA algorithm is discussed. In Section 6, Simulation Study is discussed. Results are discussed in Section 7. In Section 8 conclusions are given.

II. PROBLEM STATEMENT

To devise efficient scheduling algorithm for scheduling parallel applications represented by tasks with precedence constraints modelled by Directed Acyclic Graphs onto a network of heterogeneous processors such that all the data precedence constraints are satisfied and the overall execution time of the DAG is minimized, factoring the temporal availability of resources [11]. A new Reliable task Scheduling algorithm based on Resource Availability (RSRA) is proposed, where task scheduling decision is based on earliest finish time by considering processor availability time slot during the predicted time interval and task ready time so as to minimize makespan.

III. BACKGROUND

In this section, a brief overview of the Application Model, Cloud Resource Model and Workflow Scheduling algorithms is presented. Figure 1 shows a sample workflow application modeled by DAG scheduled by Workflow Scheduler onto Cloud resources.

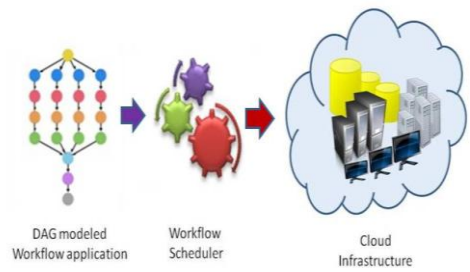

Figure 1: DAG application deployment on Cloud Resources

Application Model

Applications like Weather Forecasting, Earthquake Analysis, Bio Informatics, Astronomy, and a host of other engineering and scientific applications that require enormous computing capabilities are called High-Performance Applications. Workflows can be used to represent complex computational problems that can be efficiently processed in distributed environments. Parallel applications that consist of sub-tasks with precedence constraints can be modeled by Directed Acyclic Graphs[6][6]. The nodes represent the tasks in the parallel application and the edges represent the data transfer between tasks. Predecessor nodes are set of nodes in a DAG which have an edge directed towards a node n_i and are denoted by PRED(n_i)[12]. The set of nodes which have a directed edge from a node n_i are called its successor nodes and are denoted by SUCC(n_i). Nodes in a DAG that do not have a predecessor are called start nodes and nodes that do not have a successor node are called exit nodes. The upward rank $rank_u$ or blevel(n_i) is the bottom level of n_i and is length of the longest path from n_i to any exit node including the weight of n_i. The length of a path in a DAG is the sum of its node and edge weights. The downward rank $rank_d$ or tlevel(n_i) is the top level of n_i and is the length of the longest path from a start node to node n_i excluding the weight of n_i. The longest path in a DAG is called the critical path[6][6] [12]. A DAG may have multiple critical paths. Node weights in a DAG represent average execution times of nodes over all the processors in the target execution system. Edges represent precedence constraints between nodes. An edge (n_i,n_j) indicates that node n_j cannot start execution until ni completes execution and receives all the required data from it. The time taken to transfer data from parent task to child task is represented by edge-weights. If a DAG has multiple start nodes, then a dummy node with a zero weight is added and zero weight edges are added from the dummy start node to the multiple start nodes. Likewise, if a DAG has multiple exit nodes, a dummy exit node with zero weight and zero weight edges from multiple exit nodes to the dummy exit node are added. In a DAG, the time difference between the beginning of execution of the start node and the completion of execution of the exit node is called the makespan[12]. Figure 2 shows a sample Task DAG. Table 1 specifies the Computation cost matrix that represents the expected task execution times of each task on the various heterogeneous processors considered for task allocation.

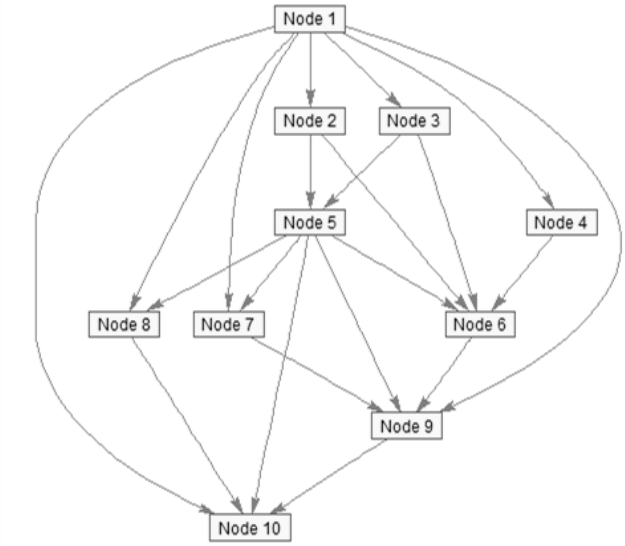

Figure 2 : Example Task Graph

© 2015-19, IJARCS All Rights Reserved

Table 1 : Computation Cost Matrix

Tasks	Resources		
	R1	R2	R3
T1	25	10	15
T2	16	22	20
T3	23	15	12
T4	10	23	12
T5	16	19	13
T6	21	18	11
T7	23	10	14
T8	13	19	10
T9	16	10	11
T10	23	14	20

Cloud Resource Model

Cloud computing is a specialized form of distributed computing that has highly extensive infrastructure that offers pools of IT resources that can be leased using a pay-for-use model[13]. Physical resources such as CPU cores, disk space and network bandwidth will be shared among virtual machines. Virtual machine characteristics include the number of CPUs, amount of memory, size of virtual disks and network bandwidth. The Virtual Infrastructure manager orchestrates resources so as to rapidly and dynamically provision resources to the applications, aggregating resources from multiple computers by continuously monitoring utilization across virtualized resource pools and reallocating available resources among VMs according to application requirements[13]. Cloud providers allow the users to select VMs based on number of cores, amount of memory and virtual images to run on machines. Varying VMs are available for a wide range of application requirements. The state of the cloud changes frequently as new VMs get added while some shut down. Service Level Agreements (SLA) specify details of services to be provided including availability and performance guarantees.

Scheduling Algorithms

Scheduling workflows is an NP complete problem. Scheduling strategies may be static, dynamic or hybrid. In Static algorithms[1], the task to VM mapping is produced in advance and executed once. The mapping is not altered through the runtime. These algorithms are extremely sensitive to execution delays and inaccurate task runtime estimation. They can adapt to the uncertainties of cloud environments, by adopting more sophisticated strategies such as conservative runtime prediction strategies, probabilistic QoS guarantees, and resource performance variations. In Dynamic algorithms[1], task to VM mapping decisions are taken at runtime. These are based on the current state of the system and the workflow execution. Once a workflow is ready for execution, scheduling decision for the first workflow task is done once at runtime. This allows them to adapt to changes in the environment so that the scheduling objectives can still be met. However, the quality of schedules produced suffers due to local task level optimizations. In Hybrid algorithms[1], there is a trade-off between the adaptability of dynamic algorithms and the performance of static algorithms.

IV. RELATED WORK

A. Heterogeneous Earliest Finish Time Algorithm

In this paper, the authors of [6] have presented two algorithms Heterogeneous Earliest Finish Time (HEFT) and Critical Path on a Processor(CPOP) for a bounded number of heterogeneous processors with an objective of simultaneously meeting high performance and fast scheduling time. The application is represented by Directed Acyclic Graph model DAG in which application tasks are represented by nodes and inter-task dependencies are represented by edges. The objective function is to allocate tasks onto processors for execution with task precedence constraints being satisfied and with minimal overall completion time. The scheduling algorithms are static in that execution times of tasks, the data size of communication between tasks, and task dependencies are known earlier. Upward rank $rank_u$ (n_i) is the length of the critical path from task n_i to the exit task, including the computation cost of task n_i.

It has two phases: Task prioritization and Processor selection. Tasks are prioritized for scheduling based on upward ranking. Tasks are arranged in decreasing order of upward rank to generate the task priority list. Tie-break is random[6] [11].

In the processor selection phase, the selected task is assigned to processor with minimum Earliest Finish Time (EFT) using insertion-based approach. There is a possible insertion of a task in an earlier idle time slot between two already scheduled tasks on a processor, however ensuring that precedence constraints are maintained.

The complexity of the HEFT algorithm is O(e x r) where ***e*** is the number of edges and ***r*** is the number of processors. For a dense graph, it is $O(v^2 \times r)$ where *v* is number of nodes and *r* is the number of processors[6][11] .

B. Expected Completion Time based Scheduling

In this paper [8], the authors propose a list scheduling heuristic for a bounded number of processors based on the expected completion time of task for heterogeneous distributed environment. The objective is to generate schedule so as minimize schedule length. The application is modelled by Directed Acyclic Graph (DAG) and the target execution system is heterogeneous distributed environment consisting of a certain number of processors in a fully connected network topology. The assumptions made were that all inter-processor communications are performed without contention; computation can be overlapped with communication and task execution of a given application is non-pre-emptive.

This algorithm has Task Prioritization and Processor selection phases. The Task Prioritization phase consists of two stages. In the first stage the priority of each task at each level is computed based on their average computation cost and maximum data arrival cost, and in the second stage, tasks are selected from all levels based on their priority. The Average Computation Cost (ACC) of a task is calculated as the ratio of sum of computation cost of the task on each processor by the number of available processors. Maximum Data Arrival Cost (MDAC) of a task is the highest amount

© 2015-19, IJARCS All Rights Reserved

of time that the task needs to spend to receive data among its parents. In the task selection stage, all tasks at each level are sorted in non-increasing order of their Expected Completion Time value, and then prioritized. The Expected Completion Time (ECT) of a task is computed by summing the average computation cost of that task and maximum data arrival cost of the same task. In the second phase, the selected task is allocated to the processor on which EFT is minimum using the insertion-based scheduling policy. The task is assigned to processor which minimizes its EFT compared to other processors. If two processors are producing same EFT for a selected task, then a processor can be selected randomly, or that which is lightly loaded, or that which is based on minimum processor utilization. The comparison metrics used for evaluation of the algorithm are Schedule Length Ratio (SLR) and Speedup.

C. Scheduling Parallel Tasks onto Opportunistically Available Cloud Resources

In this paper [14], the authors have brought out both challenges and opportunities to the problem of scheduling parallel tasks in cloud computing environment. Scientific computing, Graph processing in a large scale and Big Data MapReduce applications are resource intensive and contain tasks with synchronization requirements that spread over multiple servers. It is very difficult to schedule the backend tasks in an opportunistic manner, as the front end activities are responsible for the available slots on individual servers. On the other hand, the virtualized cloud environment allows management of tasks more effectively. Particularly, migration of the VMs of interrupted backend tasks to other available servers is possible rather than waiting indefinitely. Migration is possible at an operational cost due to the data and VM state transfer. The strategy of opportunistically scheduling tasks, has been used to share resources between jobs of different priorities while being non-intrusive to high-priority jobs. The emphasis is thus on optimizing the performance of low-priority jobs for given available resources. Low priority tasks are scheduled onto underutilized resources either by migrating or waiting. The problem is to schedule the tasks so that the combined cost due to waiting and migration is minimized. The server availability is modeled by Markov chains ON/OFF[14]. An efficient scheduling policy is proposed by the authors by formulating the problem as restless Multi-Armed Bandits (MAB) under relaxed synchronization. The scheduling policy which is based on Whittle's index reduces the complexity of the optimal policy remarkably while achieving consistently good performance under a variety of server dynamics.

D. A Monte-Carlo Approach for Full-Ahead Stochastic DAG Scheduling[4]

Just-in-time scheduling and rescheduling are two DAG scheduling approaches to deal with the unpredictability of task execution times. Just-in-time scheduling approach means that every task is scheduled only when it becomes ready i.e. when all its parent task have completed execution. Rescheduling suggests that an initial schedule for the DAG application is produced based on static prediction. Then, the allocated tasks are rescheduled according to an assessment of variations during run-time. In many cases, rescheduling can indeed improve application performance in comparison with the schedule generated based on static prediction. The predicted task execution times can be modelled in a stochastic manner. In this paper, the authors have proposed a novel Monte-Carlo based DAG scheduling[4] to generate a static schedule before run-time that minimizes the expected makespan. The input space, the set consisting of random execution time predictions of application, is defined. A set of static predictions are derived by sampling each random sample in prediction space. With a DAG and static set of associated predictions, a heuristic is applied to generate static schedule.

V. PROPOSED WORK

Reliable task Scheduling based on Resource Availability (RSRA)

We propose a new Reliable task Scheduling algorithm based on Resource Availability where task scheduling decision is based on earliest finish time of task considering the available and non-available time slots, so as to minimize makespan. In the Markov Chain Model, the probability of an event occurrence at any time is a function of the probabilities of the occurred events at previous time periods. The probabilities of resource availability can be predicted for a certain prediction interval, using the two states ON(available) and OFF(non-available) states with a 2 x 2 initial probability matrix depicting probability of 0 changing to 1 and 1 changing to 0. This can generate *n* prediction steady state vector. It may sometimes happen that a job does not complete before the start of an unavailability period of the resources. The processing of the job is therefore interrupted until the unavailability period elapses. The unavailability periods are stochastic and can occur randomly. Task execution times vary with each processor because of the variation in the CPU speeds due to heterogeneity. The estimated times and actual times vary due to delays caused by non-availability of processors. When a job is ready for execution, upon satisfying the precedence constraints and based on availability of processors, sub tasks are placed in the ready queue of allotted VMs. To improve reliability, probability of processor availability is considered in scheduling decisions.

In the proposed algorithm that has two phases, the first phase is Task Prioritization where tasks are ordered in the descending order of the upward ranks. This helps maintain the precedence constraints in the application graph. In the processor selection phase, the earliest start time of a task not only depends on the maximum available time of VM on completion of execution of previous task and task ready time but also the non-available time slots that are due to other factors such as down time, system failure, network failure, etc. The earliest finish time is the sum of earliest start time and task execution time on that VM considering the resource non available time slots. For all tasks in the priority list, the earliest start time and earliest finish time of each task on every VM is computed. The VM, which gives minimum earliest finish time is allocated for executing that task. There is a possible insertion of a task in an earliest idle time slot between two already scheduled tasks on a VM. Tasks are scheduled on the VM that gives minimum earliest finish time by taking into consideration the non-available time slots which actually introduce a certain amount of

© 2015-19, IJARCS All Rights Reserved

delay. However this increases reliability as provision is made to account for resource non-availability time slots in scheduling decisions.

BEGIN *RSRA ALGORITHM*
// *N* is set of vertices of DAG
// *P* is set of Processors
//*RdyTskLst* is the ready task list
// $ppa(1, pj)$ is probability of resource availability of processor p_j for a prediction time interval generated randomly
// $a(p_j, t)$ is resource availability matrix for processor p_j consisting of *t* timeslots generated based on ppa matrix
//*aslot*(p_j) contains starting position of next available slots on processor p_j
For all p_j in *P*
Generate random $ppa(1, pj)$
End for
For all p_j in *P*
Generate *a*(p_j,*t*)
End for
Priority List <- based on upward rank
RdyTskLst ← *StartNode*
While *RdyTskLst* is NOT NULL do
n_i ← Node in the *RdyTskLst*
For each processor p_j in *P*
Compute *aslot*(p_j)
$EST(n_i, p_j) =$
$max\ (T_{avail\,[j]}, max_{n_m \in pred\,(n_i)}(EFT(n_m, p_k) + c_{m,i}))$
If *EST(ni,pj)* < *aslot(pj)*
EST(ni,pj) = aslot(pj)
Endif
$EFT(n_i, p_j) = w_{i,j} + EST(n_i, p_j)$ + nonavailable slots
End For
Allocate node n_i on processor p_j with least EFT
Update *T_Avail*[p_j] and *RdyTskLst*
END RSRA ALGORITHM

The complexity of this algorithm is O(e x r) where ***e*** is the number of edges and ***r*** is the number of processors.

When resource availability is not considered in scheduling decisions, the makespan increases due to delays caused by resource non-available time slots. In this algorithm we consider the probability of resource availability based on which available and non-available time slots are generated for a certain prediction time interval and considered for scheduling decisions. Thus the proposed algorithm RSRA is more reliable than the existing static scheduling algorithms that do not consider resource availability. The advantage is that the scheduling objectives of minimizing makespan without increase in the resource need and reliability are met.

VI. SIMULATION STUDY

Our simulated framework first executes Random Directed A-cyclic Graph Generator Program [15] for generation of Random DAGs that model parallel applications. Graphs of varying number of nodes 10,20,30,40,50 were generated for testing the performance of the proposed algorithm, followed by execution of the proposed RSRA and algorithms specified in related work for comparison such as HEFT and ECTS. Weighted Directed Acyclic Graphs were generated with a certain number of edges generated randomly based on number of nodes given. Communication cost weighted sparse matrix was generated randomly. Heterogeneity factor [6][7][8] for processor speeds depends basically on the range percentage of computation costs on processors η = {0.1,0.5,1.0} and higher the percentage, higher the degree of heterogeneity. The average computation cost W_{DAG} of the given graph is set randomly in the graph based on the graph size. The computation cost of each task n_i on resource p_j in the system [7] is randomly from the following range:

$$W_i * \left(1 - \frac{\eta}{2}\right) \leq w_{ij} \leq W_i * \left(1 + \frac{\eta}{2}\right)$$

Resource availability probability for the given number of VMs is generated randomly. Resource availability matrix with available and non-available slots for a predicted time interval is generated randomly based on resource probability. Based on this, maximum number of resource non-available time slots is computed for all VMs for a certain prediction interval. This is randomly distributed over the prediction interval.

Performance Metrics:

The performance metrics[6] considered for the study are as follows:
(i) Makespan is the Actual Completion Time of exit node in the DAG which represents a parallel application. It is also called **Schedule Length**.
(ii) Schedule Length Ratio (SLR) is ratio of makespan to summation of minimum computation cost of critical path nodes.
(iii) Speedup is ratio of sequential execution time of parallel application to its parallel execution time. It is the ratio of minimum of summation of computation costs of tasks on processors to makespan.

The makespan was determined for the proposed algorithm RSRA and then other performance metrics Schedule Length Ratio and Speedup were computed based on schedules. The delay for HEFT and ECTS schedules based on the resource availability time slots matrix for the predicted time interval was estimated. These performance metrics were compared with HEFT and ECTS algorithms simultaneously for the same set of experiments.

VII. RESULTS AND DISCUSSIONS

The comparative results of the proposed algorithm RSRA with HEFT and ECTS are presented. Randomly generated DAGs of sizes 10, 20, 30, 40 and 50 nodes were generated and used to test the performance of RSPA algorithm, while comparing with HEFT and ECTS. The RSRA algorithm was also tested with real life application scientific workflows in the area of Astronomy and Bio-Informatics such as Montage and Epigenomics. The performance of RSRA was studied for parallelism and speedup by executing a random graph of 50 nodes for varied number of processors 9, 12 and 15 processors. The algorithms were executed to determine the performance metrics of Average Makespan, Average Schedule Length Ratio and Average Speedup for RSRA,HEFT and ECTS algorithms.
(i) Test Case 1

© 2015-19, IJARCS All Rights Reserved

Here RSRA algorithm performance is compared with HEFT and ECTS algorithm on the basis of various graph sizes. Since HEFT is robust, has low running time, and is tested to give stable performance over wide range of graph structures[6], it is considered for comparison. ECTS algorithm performs better than HEFT and employs level sorting in task prioritization phase and hence is also considered for performance comparison. Randomly generated DAGs of sizes 10,20,30,40 and 50 nodes are considered for performance testing of algorithms. For each random DAG generated, our algorithm was executed 200 times with different probability of processor availability based on which processor availability matrix was generated for a certain prediction interval. The probable delay due to processor non-availability for HEFT and ECTS were determined. From the results, we observe makespan of RSRA is better that HEFT and ECTS in this scenario which considers processor availability factor. The proposed RSRA algorithm is reliable as the resource availability factor is incorporated in our scheduling decisions. Comparative results for the performance metrics Makespan, Schedule Length Ratio and Speedup are shown below. Figure 3 shows the comparison of RSPA with HEFT and ECTS based on Average Makespan.

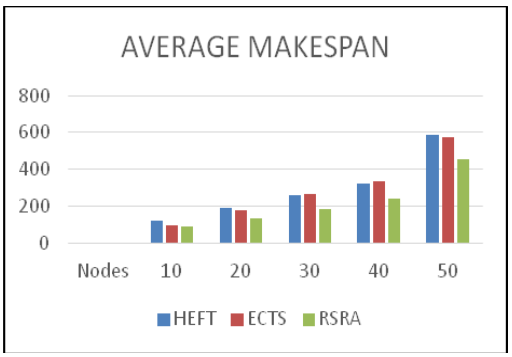

Figure 3 : Comparison of RSRA with HEFT and ECTS based on Average Makespan

Figure 4 shows comparison of RSRA with HEFT and ECTS on the basis of Average SLR. The lower the SLR, the better the performance. It is seen from the figure that RSPA performs better than HEFT and ECTS.

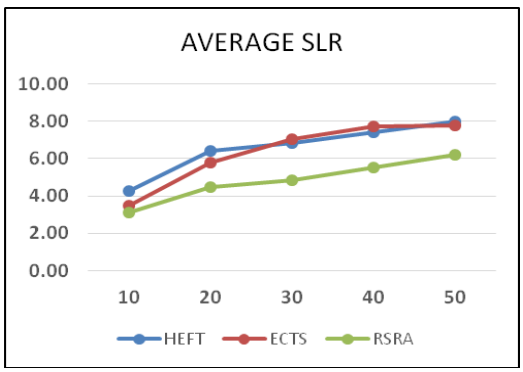

Figure 4: Comparison of RSRA with HEFT and ECTS based on Average SLR

Figure 5 shows comparison of RSRA with HEFT and ECTS on the basis of Average Speedup. The higher the speedup, the better the performance. It is seen from the graph that RSPA performs better that HEFT and ECTS.

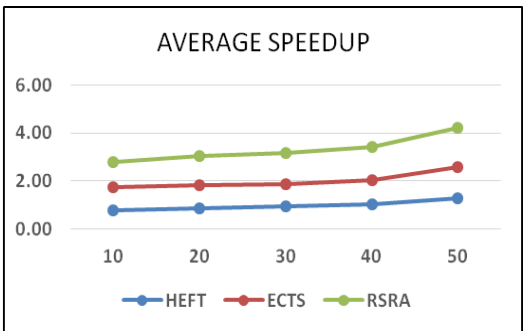

Figure 5: Comparison of RSRA with HEFT and ECTS based on Average Speedup

(ii)Test Case 2

Here, the proposed algorithm is tested for performance on scientific workflow application graphs such as Montage and Epigenomics. Montage workflow[1] created by NASA/IPAC is an astronomy application characterized by being I/O intensive that is used to create custom mosaics of the sky based on a set of input images. A composite image of a region of the sky that is too large to be taken by astronomical camera or images that have been measured with different wavelengths and instruments can be generated. During the workflow execution, the geometry of the output image is calculated from that of the input images. Afterwards, the input data is re-projected so that they have the same spatial scale and rotation. This is followed by a standardization of the background of all images. Finally, all the processed input images are merged to create the final mosaic of the sky region. The structure of this workflow is shown in Figure 4. The Epigenomics[1] workflow of Bioinformatics field, is a CPU intensive application that automates the execution of various genome sequencing operations. This workflow is created by the USC Epigenome Center and the Pegasus Team. It is used to automate various operations in genome sequence processing. Figure 6 shows a sample Montage Workflow.

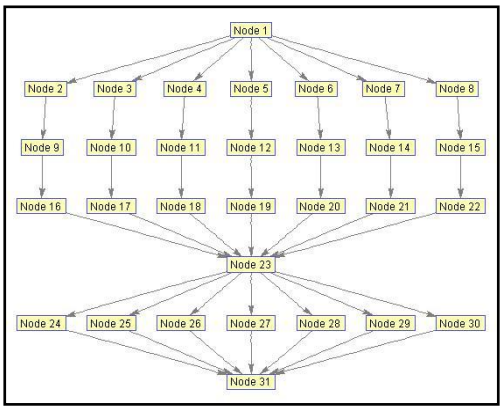

Figure 6 : Sample Montage Workflow

Figure 7 shows the comparison of RSRA with HEFT and ECTS based on Average Makespan.

© 2015-19, IJARCS All Rights Reserved

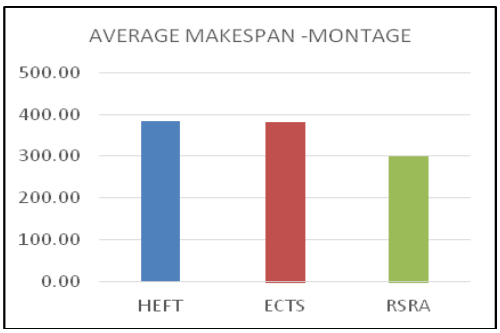

Figure 7 : Comparison of RSRA with HEFT and ECTS based on Average Makespan

Figure 8 and 9 show comparison of the proposed algorithm with HEFT and ECTS based on Average SLR and Average Speedup respectively.

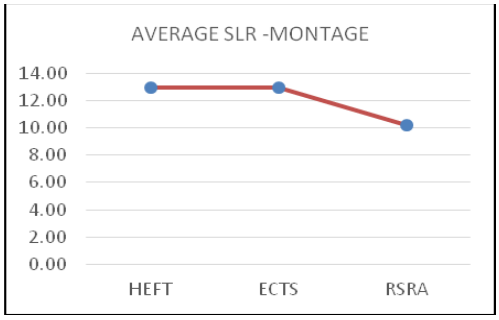

Figure 8:Comparison of RSRA with HEFT and ECTS based on Average SLR

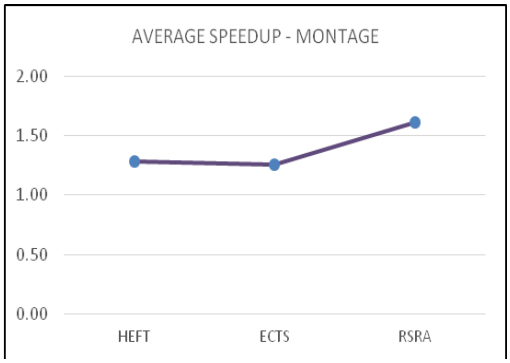

Figure 9: Comparison of RSRA with HEFT and ECTS based on Average Speedup

Figure 10 shows a sample Epigenomics Workflow used in Bioinformatics. Figures 11,12,13 compare the proposed RSRA algorithm with HEFT and ECTS algorithms based on Average Makespan, Average SLR and Average Speedup

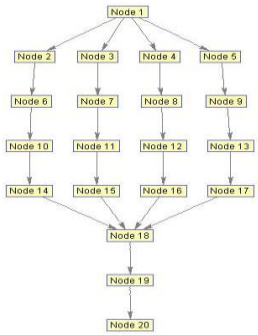

Figure 10 :Sample Epigenomics Workflow

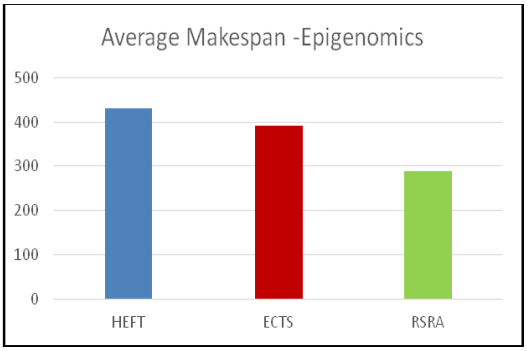

Figure 11:Comparison of RSRA with HEFT and ECTS based on Average Makespan

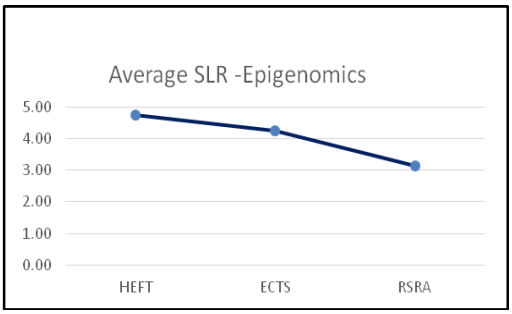

Figure 12:Comparison of RSRA with HEFT and ECTS based on Average SLR

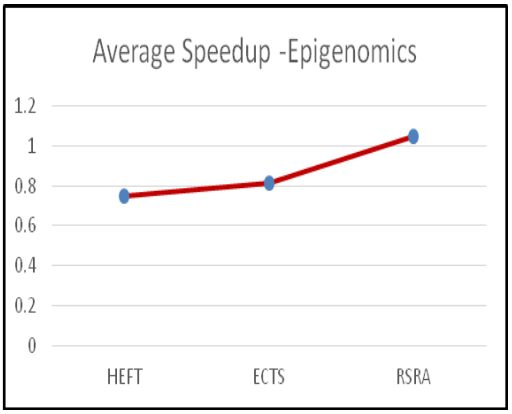

Figure 13: Comparison of RSRA with HEFT and ECTS based on Average Speedup

(iv) Test Case 3

Here we study the performance of RSRA algorithm for parallelism, considering a large 50 node random directed acyclic graph being executed with various number of processors such as 9,12,15. Figure 14 shows that average

© 2015-19, IJARCS All Rights Reserved

makespan decreases with increase in the number of VMs used. As the number of VMs increase which indicates higher parallelism, the makespan reduces.

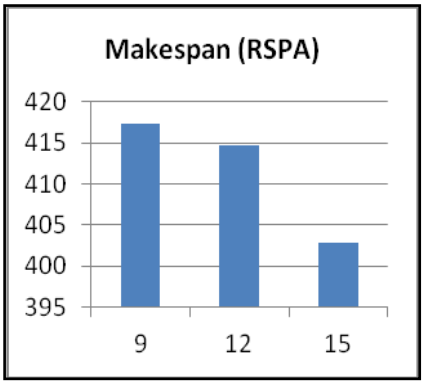

Figure 14: Average Makespan of 50 node random graph for varying number of processors

VIII. CONCLUSIONS

Virtual machines provided by cloud infrastructure do not exhibit a stable performance[1] due to network connectivity as well as computational nodes and may have a significant impact while scheduling workflows on clouds[1].Since the resources are not dedicated and can be used by other users simultaneously, there are load variations on resources, resulting in fluctuations in resource availability that affects the schedules. In this paper, we propose our new algorithm Reliable task Scheduling Algorithm (RSPA) based on Processor Availability for scheduling application graph on to heterogeneous computing environment. Our algorithm performs better than HEFT algorithm in a scenario where resource availability factor is considered. We have presented four test cases, in the comparative experimental study of performance of RSPA. When algorithms such as RSPA,ECTS, HEFT were executed, in an environment considering resource availability factor, the performance metrics such as makespan, SLR and speedup were found to be better in RSPA than in others, for random graphs with varying sizes and real life application graphs Montage and Epigenomics in the areas of Astronomy and Bio-Informatics. For various sizes of graphs with 10,20,30,40,50 nodes, the performance improvement of RSPA when compared with HEFT, is ranging between 35% and 43%. When RSPA is compared with HEFT, on an average, the overall percentage reduction in makespan for graphs of various sizes is 37%. When RSPA is compared with ECTS algorithm, the percentage reduction in makespan ranges from 11% to 45% for various sizes of graphs. On an average, the overall percentage reduction in makespan for graphs of various sizes when RSPA is compared with ECTS is 30%. Also there is considerable percentage reduction in makespan for real life application graphs such as Montage and Epigenomics. The simulation results show that RSPA algorithm performs better and is more reliable in environment where resource availability is continuously varying than other algorithms that assume full processor availability while scheduling tasks.

REFERENCES

1] M. A. Rodriguez and R. Buyya, "A Taxonomy and Survey on Scheduling Algorithms for Scientific Workflows in IaaS Cloud Computing Environments", CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE, Concurrency Computat.: Pract. Exper. 0000; 00:1–32,Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.4041

2] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds." IEEE Transactions on Cloud Computing, Volume 2, Issue 2, Pages: 222-235, 2014.

3] vSphere Resource Management ESXi 6.5 vCenter Server 6.5 Documentation, VMware

4] Wei Zheng, Rizos Sakellariou," A Monte-Carlo Approach for Full-Ahead Stochastic DAG Scheduling",**2012** IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

5] Brent Rood and Michael J. Lewis, "Grid Resource Availability Prediction-Based Scheduling and Task Replication," Journal of Grid Computing, 2009.

6]Haluk Topcuoglu, Salim Hariri and Min-You Wu, "Performance Effective and Low Complexity Task Scheduling for Heterogeneous Computing", IEEE Transactions on Parallel and Distributed Systems, Vol.13, No.3, March 2002, : 260-274

7] E. Illavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments", January 2007, Journal of Computer Science[Online]. 3(2). pp. 94-103.

8] R. Eswari and S. Nickholas, " A Level-wise Priority Based Task Scheduling for Heterogeneous Systems", International Journal of Information and Education Technology, Vol. 1, No. 5, December 2011, p.p. 371-376 ISSN: 2010-3689,DOI: 10.7763/IJIET.2011.V1.60 (ECTS)

9] Deepak Poola, Saurabh Kumar Garg, Rajkumar Buyya, Yun Yang, and Kotagiri Ramamohanarao,"Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds", 2014 IEEE 28th International Conference on Advanced Information Networking and Applications

10] Mustafa M. Al-Sayed, Sherif Khattab, Fatma A. Omara," Prediction mechanisms for monitoring state of cloud resources using Markov chain model", Journal of Parallel Distributed Computing 96 (2016) 163–171,Elsevier

11] Chitra S and Prashanth C. S. R, "Probabilistic Availability based Task Scheduling Algorithm," 2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15), Bangalore, 2015, pp. 1-4.doi: 10.1109/ITACT.2015.7492649

12] S. Baskiyar and P. C. SaiRanga, "Scheduling directed a-cyclic task graphs on heterogeneous network of workstations to minimize schedule length," 2003 International Conference on Parallel Processing Workshops, 2003. Proceedings., 2003, pp. 97-103. doi: 10.1109/ICPPW.2003.1240359

13] Buyya R, Broberg J, Goscinski A,"Cloud Computing: Principles and Paradigms", Wiley Publications

© 2015-19, IJARCS All Rights Reserved

14] T. He, S. Chen, H. Kim, L. Tong and K. W. Lee, "Scheduling Parallel Tasks onto Opportunistically Available Cloud Resources," 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, 2012, pp. 180-187,doi: 10.1109/CLOUD.2012.15

15] Yinfeng Wang, Zhijing Liu, Wei Yan, "Algorithms for Random Adjacency Matrixes Generation Used for Scheduling Algorithms Test", International Conference on Machine Vision and Human-Machine Interface (MVHI), 2010

© 2015-19, IJARCS All Rights Reserved