Response-Aware Scheduling of Big Data Applications in Cloud Environments

Sambit Kumar Mishra, P. Satya Manikyam, Bibhudatta Sahoo, Mohammad S. Obaidat, Fellow of IEEE & Fellow of SCS, Deepak Puthal and Mahardhika Pratama
National Institute of Technology, Rourkela, India
Fordham University, USA and University of Jordan, Jordan
University of Technology Sydney, Australia
Nanyang Technological University, Singapore
Email: m.s.obaidat@ieee.org and Deepak.Puthal@uts.edu.au

***Abstract*—The cloud infrastructures afford a proper environment for the execution of large-scale big data application. The scheduling of a substantial number of tasks in the heterogeneous multi-tenant cloud environment is one of the most significant research challenges in the current era. The major challenges of task allocation are to optimize the overall completion time, cost of execution, tardiness and utilize idle resources of cloud effectively. In this paper, we have proposed a novel scheduling algorithm for task allocation of the cloud resources to optimize the overall execution time by minimizing response time. In order to find the effectiveness of our proposed algorithm, we have compared our solution with six standard competing algorithms for the optimization of performance metrics in the cloud environment. The results confirm that our proposed algorithm operates better than the other state-of-the-art algorithms in terms of response time (allocation time), makespan and the total execution time.**

***Keywords*—Scheduling; cloud computing; task allocation; allocation time; execution time**

I. Introduction

Cloud Computing is a service delivery model that affords computing resources on demand from data centers to applications over the Internet on the pay-for-use basis. The primary intent of using this technology is to increase efficiency, performance, and to decrease the cost [1]. The physical cloud resources are virtualized to serve a large number of users at a time. This is possible because of the virtualization technique used in the cloud system. The technology behind virtualization is virtual machine monitor (VMM) or hypervisor that separates the computer environments from the actual physical infrastructure [23], [24]. The issue with this model is: 'how to allocate resources and jobs by satisfying features of cloud computing?' The efficient scheduling algorithm is needed to schedule the resources to jobs by meeting certain Quality of Services (QoS) [2]. There has been a lot of different task scheduling algorithms which have their advantages and disadvantages. An incompatible scheduling of tasks may result in inefficient utilization of the resource. Optimal scheduling will schedule a maximum number of jobs to a minimum number of resources while meeting the QoS which leads to decrease in cost and response time [20].

In the modern time, a large number of cloud users interact with the CSP simultaneously for getting services. These huge number of varied size tasks with their required data are stored in Big Data which mainly provides Storage-as-a-Service (SaaS) properties. The Cloud Service Provider (CSP) renders a reliable and scalable environment for the big data applications through the cloud computing system. The business model is one of the important issues in Big Data [3]. This paper addresses methods and environments to accomplish Cloud services for Big Data applications. The compelling combination of Cloud and Big Data enables the on-demand storage, space, and computing power for the cloud users [21], [22]. A Big Data cloud incorporates an extremely scalable, efficient, and a low-cost data storage platform [4].

The basic standard optimum algorithms for task scheduling are min-min, and max-min and the remaining algorithms are the modification of these algorithms. Based on research work [5], [6], [7], it had declared that suffrage algorithm is better in terms of makespan. There are some algorithms which combine both advantages of min-min and max-min to minimize the starvation of tasks and makespan [8], and some scheduling algorithms combine the min-min and sufferage for better task allocation [9]. However all existing algorithms had better performance, but a high range of diversity and inconsistency of real-time tasks may degrade their performance in real environments. By considering all these issues in the standard algorithms, a new algorithm has been designed which try to minimize the makespan of the system with minimum starvation and execution time of the algorithm. The literature survey is indicating that the proposed algorithm is a new method which is not following the characteristics of existing algorithms. This method is not a combination of any standard algorithms, and it follows its own procedure to reduce makespan.

Scheduling is defined as a set of policies to manage the flow of work which will be executed by computing resources [10]. In the cloud computing, task scheduling can be defined as a set of policies, and factors that determine and choose the task from the task set to execute on a set of minimum available resources at a particular instant of time. So, the dominant component of the cloud system performance solely depends on task scheduling algorithms. Task scheduling algorithms have to achieve high performance and efficient system throughput. The good scheduling algorithm has to use a minimum number of resources while maintaining the SLAs. There are different types of scheduling algorithms based on different policies. Some policies try to minimize the makespan, load, energy etc.

Most of the algorithms in the literature focused on minimizing the makespan. In the cloud environment, user submits tasks to the data center broker or cloud service provider (CSP). The broker will submit these tasks to VMS. The broker is an intermediary between cloud and user. In this paper, a new algorithm has proposed that is more efficient in execution time than existing task scheduling algorithms with comparable makespan.

Definition I.1. Makespan is the time taken to complete the execution of all tasks after submitting them for scheduling.

Definition I.2. Allocation time is the time required to take the decision by the algorithm to map tasks to virtual machines for optimal scheduling.

Definition I.3. Total Time is the sum of Makespan and Allocation time for this work.

Definition I.4. Expected time to compute (ETC) tells the possible execution time of a given task on a given Virtual Machine. The ETC Matrix shows execution time of each task on every VM in matrix format.

Definition I.5. Expected completion time (ECT) is the probable completion time of a task on assigned Virtual machine. It is the addition of Virtual Machine waiting time and ETC of the given task.

The remaining of the paper is constructed as follows. Section II outlines a summary of related work regarding the task scheduling in cloud computing; Section III describes a brief idea about the problem statement with the description of the system model (includes host model, VM model, and task model). Section IV describes our proposed algorithm to minimize the makespan of the heterogeneous computing environment along with an explanation of an example. Section V, explains simulation model and results, where the effectiveness of our algorithm is shown followed by the conclusion of the paper in Section VI.

II. Related Work

The scheduling of Big Data tasks is an assignment problem where tasks have to map to the cloud virtualized resources. Assignment problem is a well-known NP-hard problem. A huge number of researchers have been working on this issue, and they have proposed various heuristic techniques for it [13], [14]. Due to a large amount of interactive data or information and need to a deadline to execute services, usually, in most of the cases, users could not get the service with acceptable QoS. After receiving the service request from the user, the resources (e.g. CPU, network bandwidth, main memory, secondary storage, etc.) of the cloud data center are virtualized. Task execution time is more important to everyone, and for this, there should be a proper management of physical resources by an efficient mapping between the tasks and VMs.

Some proposed works related to this paper are explained in TABLE I. All the existing algorithms concentrated primarily on reducing the makespan of the system. Out of them, some works have been done with multi-objective solutions. Researchers have used First Come First Serve (FCFS) technique to allocate tasks to VMs with makespan minimization as the objective [11]. In their model, the smaller task has to wait if a larger task is in front of them in the queue which degrades the makespan. In [12], authors have used Opportunistic Load Balancing (OLB) technique where they mainly focused on load balancing through makespan of the cloud system. In TABLE I, the last column shows the time complexity of different algorithms, where n is the number of input tasks and m is the number of VMs.

The Min-Min algorithm calculates minimum completion time of all the tasks [15]. Then, it chooses and assigns the task with minimum completion time. There is a chance of starvation in most of the scheduling problem in the cloud environment, because of a huge number of tasks. The limitation of the Min-Min algorithm is starvation of larger task [15]. The similar procedure followed by Max-Min [15], there it leads to starvation of smaller tasks. Some existing works [10], [8], [9] used both approaches to take the advantage of min-min and max-min to get less makespan. There has been no existing work which focuses on reducing makespan, allocation time, and starvation of tasks parallely. To achieve all these objectives, in this paper, we have proposed a new algorithm.

III. Problem Statement

The scheduling of Big Data task in the cloud environment is a multi-objective problem. The objectives can be 1) minimizing makespan; 2) avoiding starvation; 3) load balancing; 4) minimizing allocation time; 5) optimizing resource utilization; 6) maintaining certain SLAs; 7) Guarantee QoS. In this paper, we concentrated on the minimizing the makespan, waiting time, and most importantly the allocation time. The model about the system gave a brief idea about characteristics of physical machines and their resources, virtual machines initiated on them.

The system model for the scheduling of task in the cloud environment is shown in Fig. 1. The cloud users submit their tasks through various sources (i.e., sources can be mobiles, computers, etc.). Sources generate tasks and submit these tasks to cloud service provider for execution. The generated tasks are stored in the task pool. The task scheduler (CSP) mapped the input tasks to VMs. The VMs are heterogeneous in terms of resource capacity and hosted on different physical machines. Every physical machine can host multiple number of VMs through virtualization technique. At the time scheduling, the CSP allocates tasks to VMs without considering host. By considering this fact we denoted Virtual machine as V_{ij} with respect to host and VM_k with respect to scheduling where i is host number, j is the VM number on that host and k is the total number of VMs available in the cloud system. The proposed model is a generalized model and can be applied to different scenarios depending on the requirement of the application. Mainly, the host (physical machine) contains resources required to compute the task like resources for storage, computation, network, etc.

Let $\{t_1, t_2, t_3, ..., t_n\}$ be the set of tasks and $\{V_1, V_2, V_3, ..., V_m\}$ be set of VMs. Assign the tasks to VM in such a way that

$$Min(\sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} f(v_i, t_j))$$

Subjected to the constraints $X_{ij} \neq X_{kj}$.

$$Min(\sum_{i=1}^{n} W_i)$$

TABLE I. SOME ALGORITHM DESCRIPTIONS WITH THEIR LIMITATIONS AND TIME COMPLEXITY

Algorithm	Description	Disadvantages	Time Complexity
Sufferage [10]	It computes the difference between earliest expected completion time and second earliest completion time for each task. This is called as suffrage value of task. Task which is having high suffrage value will be assigned first and that task will be removed. Same process repeats for remaining tasks.	The allocation time of this algorithm is high. If there are more tasks with same suffrage value then it always selects the first one which leads to starvation of remaining tasks.	$O(mn^2)$
FCFS [11]	It allocates the tasks to resources according to their arrival time. Task in front of queue will be assigned to available resources.	Waiting time of tasks is more. Small tasks have to wait, if larger tasks are in front them in the queue, which leads to high waiting time.	O(mn)
Round Robin [11]	In this algorithm scheduler assigns a fixed time quantum to each process and it circulates among tasks in round robin manner.	Context Switching burden is there. It is not optimum as compared to other algorithms.	O(n)
OLB [12]	This algorithm mainly focuses on balancing the load among VM. It randomly selects an task and assigns it available resource. The concern is on keeping resources busy all the time.	Poor makespan	O(mn)
Min-Min [15]	In this algorithm minimum expected completion time for every task is calculated . From this set minimum of minimum expected completion time of all tasks is calculated and that corresponding task is assigned to virtual machine. This task is removed from the set. Same procedure is followed for remaining tasks.	According to min-min strategy small tasks assigned first so larger tasks waiting time is more.	$O(mn^2)$
Max-Min [15]	In this algorithm minimum expected completion time for every task is calculated . From this set, maximum of minimum expected completion time of all tasks is calculated and that corresponding task is assigned to virtual machine. This task is removed from the set. Same procedure followed for remaining tasks.	According to max-min strategy larger tasks are assigned first so small tasks waiting time is more.	$O(mn^2)$
RASA [8]	Resource Aware Scheduling Algorithm is combination of min-min and max-Min. It tries to overcome the disadvantages of these algorithm and simultaneously taking their advantages also. The main step is that it alternates between min-min and max-min for every iterations. If number of tasks are even it follows min-min otherwise it follows max-min algorithm.	Moderate Makespan & High allocation time	$O(mn^2)$
TASA [9]	TASA (Task-aware scheduling algorithm) combines both Min-min and Suffrage algorithms. It alternatively applies suffrage and Min-Min strategies, if the number of tasks are even and odd respectively.The allocation strategy flips randomly. TASA mainly concerns about the number of tasks rather than resources available.	Even though it alternates between max-min and sufferage if two maximum tasks have same sufferage value then first task will be assign while second task has to wait. The allocation time of this algorithm is high.	$O(mn^2)$
Selective [16]	This algorithms found the standard deviation of tasks. If half of tasks size is greater than standard deviation then it will assign tasks by min-min otherwise by max-min.	Always not guarantee minimum makespan	$O(mn^2)$

$$Min(\sum_{i=1}^{n} C_i)$$

Where $f(v_i, t_j))$ will return time taken to compute t_j on v_i.W_i is waiting time of the tasks.$\sum_{i=1}^{n} C_i$ is number of primitive steps taken by scheduling algorithm.

A. Host Model

We consider a virtualized cloud which contains a set $H = \{h_1, h_2, ...h_l\}$ of physical computing hosts, to provide hardware infrastructure for virtualized resources and size of the set is l which is finite. Each host $h_i, 1 \leq i \leq l$ contains following parameters.

$$h_i = \{hId_i, hTRes_i, hFRes_i, hVM_i\}.$$

where,

- hId_i : identification number of i^{th} host.
- $hTRes_i = \{hTR_{i1}, hTR_{i2}, ..., hTR_{ik}\}$ such that $hTRij, i \in [1, m], j \in [1, k]$ is the total resource capability of j^{th} resource running on i^{th} host.
- $hFRes_i = \{hFR_{i1}, hFR_{i2}, ..., hFR_{ik}\}$ such that $hFRes_{ij}$, $i \in [1, m], j \in [1, k]$ is the free resource capability of j^{th} resource running on i^{th} host.
- hVM_i : set of virtual machines that are running on i^{th} host.

For each host $h_i \in h$, it contains a set of active virtual machines running on the host indicated by set V_i such that $V_i = \{V_{i1}, V_{i2}, ..., V_{i|V_i|}\}$ of virtual machines and each VM V_{ij} has processing capability p_{ij} that is subject to the constraint

$$\sum_{j=1}^{|V_i|} p_{ij} \leq p_i$$

where p_i is processing capability of i^{th} host.

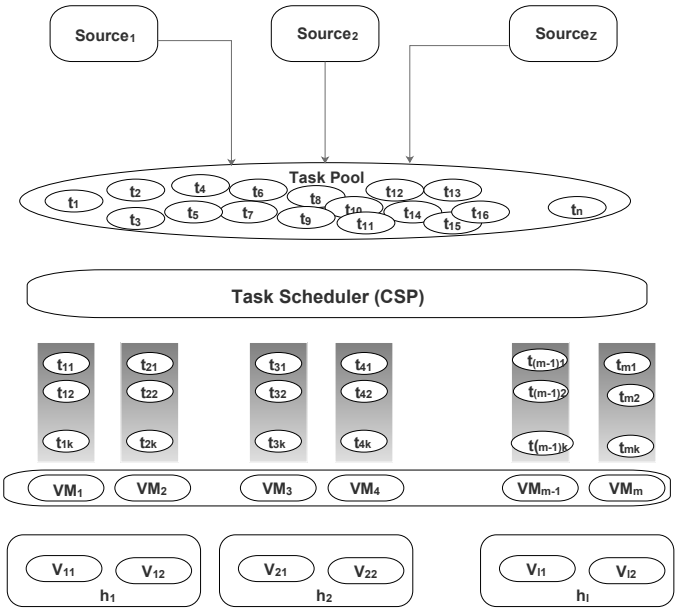

Fig. 1. Cloud system model.

B. Virtual Machine Model

We consider VMs are heterogenous in terms of resource capacity. The set $VM = \{VM_1, VM_2, ...VM_m\}$ is the set of finite number of VMs. Each VM is described as follows.

$$VM_j = \{VMId_j, VMp_j, tId_j, tque_j, hId_j\}.$$

- $VMId_j$: identification number of j^{th} virtual machine.
- VMp_j : processing power of the VM.
- tId_j : identification number of currently executing task.
- $tque_j$: queue of waiting tasks assigned to this VM. This Queue contains identification number of waiting tasks.
- hId_j : identification number of host where this VM is running.

C. Task Model

Let $T = \{t_1, t_2, ...t_n\}$ be set of tasks that came for execution on to the cloud. Each task can contains following parameters:

$$t = \{tId_k, tSz_k, tSta_k, tVM_k\}$$

each parameter is defined as follows:

- tId_k : Task identification number.
- tSz_k : Size of task.
- $tSta_k$: Status of task indicating whether task is executing, waiting, completed, allocated, etc.
- tVM_k : Identification number of VM to which this task is allocated.

IV. Proposed Algorithm

The proposed algorithm will take all user tasks and VMs as input. It will create the maximum priority queue for tasks based on their size and VMs based on their processing capability. The algorithm tries to schedule the tasks from both sides of the task priority queue. The first iteration will start from the first task in the priority queue which is the largest task. It assigns this task to a virtual machine which computes it in a minimum amount of time. Let's say it as $mintime$. Now in this minimum amount of time, it will try to assign as many numbers of small tasks as possible from the rear of the queue. It will remove all assigned tasks from the task set. Next iteration again it will check from starting of priority queue for task scheduling. Same process repeats. To reduce time complexity of the algorithm, we maintained the special list for VM called it as $VMWtlist$. At the time of assigning task to VM, instead of updating ETC matrix, we updated VM waiting time. So, always ETC contains expected time to execute rather than expected time to completion of the task. This saves a lot of time since ETC matrix size is an order of $no\_task * no\_VM$, its updation time will be in order of a number of tasks whereas updating VM will take only constant time. To compute the minimum completion time of a task, first a list will be calculated which is a sum of VMs waiting time and tasks expected time for completion. The minimum value of this list will be returned as minimum completion time. Since the proposed algorithm assigns the task in parallel from top to bottom of the queue, larger or smaller tasks would not wait for a large amount of time.

Algorithm 1

Input: $task\_list$,VM_list
Output: MakeSpan.

1: Build priority queue of VMs in decreasing order of their processing speed
2: Build priority queue of tasks in decreasing order of their Size
3: VM_Wtlist initialize to zero. $till\_time = 0$;
4: $from\_first = 0$;
5: $from\_last = no\_tasks$;
6: Compute ETC Matrix;
7: **while** $from\_first <= from\_last$ **do**
8: $[min\_time, VMind] = MCT(task, from\_first)$;
9: $Update(VMind, min\_time)$;
10: $till\_time = Max(till\_time, min\_time)$;
11: **while** $MCT(task, from\_last, VM\_Wtlist) < tillItime$ **do**
12: $[min\_time1, VMind1] = MCT(task, from\_last, VM\_Wtlist)$;
13: $Update(VMind1, min\_time1)$;
14: $from\_last = from\_last - 1$;
15: **end while**
16: $from\_first = from\_first + 1$;
17: **end while**

The step-wise explanation of algorithm is as follows. We are creating two priority queues in the beginning, for VMs (high priority to VM with faster processing speed) and tasks (high priority to the largest task) respectively. So, we are able to select VM, task in O(1) time. We have also created one array called Vm_Wtlist to maintain the availability time of virtual

machines in constant time. The size of this array is equal to the number of virtual machines. Since we start allocating tasks from both sides of the queue, we maintained two indexes called $from\_first$ and $from\_last$. $from\_first$ indicating the task index from the starting. Task with index less than $from\_first$ has been assigned . In the same way, $from\_last$ will also work. Here while selecting a task or VM from the queue, it may not give minimum value since the priority queue had given priority to a large sized task. We moved towards the allocation considering the difference between global minimum and popped element size is negligible . We computed Expected Time to Compute matrix in step 6. Now, we started the first task in the priority queue and found out its minimum completion time in step 8 using the function call minimum completion time indicated by MCT ($Algorithm$ 3). It will return min_time and corresponding VM index where to allocate the task. We updated the VM waiting time using the VM Index through the Update function ($Algorithm$ 2). Now, within this execution time min_time, we assigned as many numbers of as smaller sized tasks as possible from bottom side of queue using $from\_last$ index from step 11-15. The variable $till\_time$ contains the makespan. It will update every time in step 16. The entire process repeats till all the tasks have assigned. Since at each and every step, we are reducing the allocation time of algorithms, the total allocation time is very less compared to other algorithms. We are following optimum criteria for task scheduling to get better makespan as compared to other algorithms. Since tasks are assigned from both the sides of the queue in parallel, the starvation of tasks is lower as compare to remaining algorithms.

Algorithm 2 Update

Input: VM_ind,wt_time
Output: Updated VM waiting list
1: $VM\_wtlist[VM_ind] = VM\_wtlist + wt_time$

Algorithm 3 Minimum Completion time Algorithm(MCT)

Input: $(taskind, ETC,)$
Output: min_time, VM_ind
1: $i = 0$
2: **while** $i < no\_VM$ **do**
3: $task\_etc(i) = etc(i, :) + VM\_wtlist(i)$
4: **end while**
5: [min_time,VM_ind]= $min(task\_etc)$;

- Line 1, and Line 2 will take $O(n)$, $O(m)$ to create priority queue, respectively.
- Line 6 will take $O(m \times n)$ time.
- Line 7 will take at most $O(n)$ time in worst case. The task taken through $from\_last$ will take O(1) time since the $from\_last$ may not be global minimum task.
- line 11 MCT will take $O(m)$ time.
- Total time complexity is $O(m \times n)$.

A. Algorithm Explanation with Example

To explain the algorithm through an example, we have considered the following scenario. There are three VMs $\{V1, V2, V3\}$ with processing powers of $\{2000, 1000, 500\}$ respectively. Let $\{T1, T2, T3, T4\}$ be the tasks with sizes in MIPS as $\{10000, 8000, 4000, 2000\}$, respectively. The queues will be formed for VMs and Tasks in O(v) and O(t) time, respectively. Every VM contains one attribute to indicate its waiting time means after how much time it can process another task. Initially, the waiting time of every VM is zero. Now, the large sized task is assigned to high processing power capable VM. So, $T1$ is assigned to $V1$. The waiting time of $V1$ is updated as 8. The local makespan is updated as 8. Within this time, the algorithm allocates as many numbers of tasks as possible from the bottom of the queue. It keeps on allocating lower sized tasks to VMs which have less waiting time. So, $T4$ and $T3$ are assigned to $V2$ within the makespan of 8. It is not possible to allocate $T2$ to any VM with makespan less than 8. So, Algorithm goes to above and starts continuing the process from $T2$. It repeats if the number of tasks is higher. The largest task assigned to VM with faster processing speed implies the local makespan is lower, and within the time, many tasks are allocated to reduce the global makespan. Since large tasks and small tasks are assigned from both ends, the starvation will be less during the execution of tasks. This way will results in less makespan with less starvation and allocation time.

TABLE II. TASK-DESCRIPTION

Task-id	T1	T2	T3	T4
Task-Size (MI)	16000	8000	4000	2000

TABLE III. VM-DESCRIPTION

VM-id	V1	V2	V3
VM-Speed (MIPS)	2000	1000	500

$$ETC - Matrix = \begin{bmatrix} 8 & 16 & 32 \\ 4 & 8 & 16 \\ 2 & 4 & 8 \\ 1 & 2 & 4 \end{bmatrix}$$

VM Waiting	time	0	0	0	VM Waiting	Time	8	2	0
1. For	T1 min T1	8 8 V1	16	32	3.For	T3 Min T3	10	6 6<8 V2	8
MakeSpan		8			VM Waiting	Time	8	6	0
VM Waiting	Time	8	0	0	4.For	T2 Min	12 12>8	14	16
2. For	T4 Min T4	9	2 2<8 V2	4	Don't	Assign	Go	To	Top
					For	T2 Min	12 12	14	16
VM Waiting	Time	8	2	0	MakeSpan		12		

V. SIMULATIONS AND RESULTS

We evaluated the new proposed algorithm through simulation with generated datasets. The experiments were done using MATLAB R2014a version on Intel Core 2 Duo processor, 2, 20GHZ CPU and 4GB RAM running on Microsoft Windows 8 platform. We note that MATLAB simulator has been broadly approved to evaluate schemes suggested in the literature

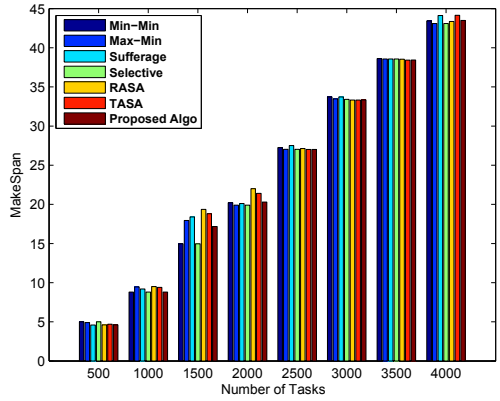

Fig. 2. Comparison of Makespan of the system for different algorithms for small sized tasks.

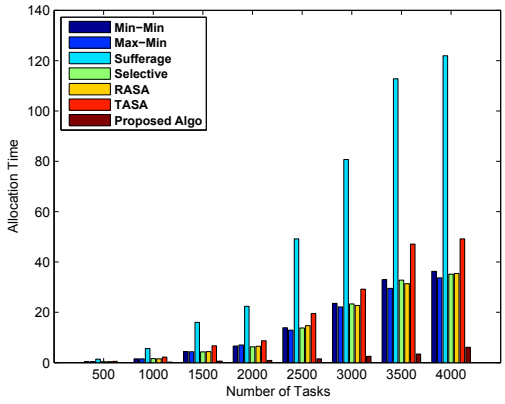

Fig. 3. Comparison of allocation time of the system for different algorithms for small sized tasks.

[17], [18]. The task arrival rate is generated with random distribution. We have simulated and compared our proposed algorithms with remaining algorithms in two scenarios as follows:

- **Scenario-1** We are fixing the number of VMs at 100. The size of tasks and capacity of VMs is in between 1000-2000 Million Instructions (MI) and 1000-2000 Million Instructions Per Second(MIPS), respectively. In our simulation, the number of tasks is varied from 500 to 4000 in intervals of 500.

- **Scenario-2** We are fixing the number of VMs at 500. The size of tasks and capacity of VMs are in between 10000-20000 Million Instructions (MI) and 1000-2000 Million Instructions Per Second(MIPS), respectively. For simulation, the number of tasks are varied from 500 to 4000 in intervals of 500.

We tabulated the makespan and execution time of algorithms under above scenarios. Every experiment is conducted ten times for different input values, and the average of their results are listed as shown in Tables II and III. Tables IV and V show the comparison of various parameters (Makespan (MS), Allocation Time (AT), Total Time (TT)) for our proposed

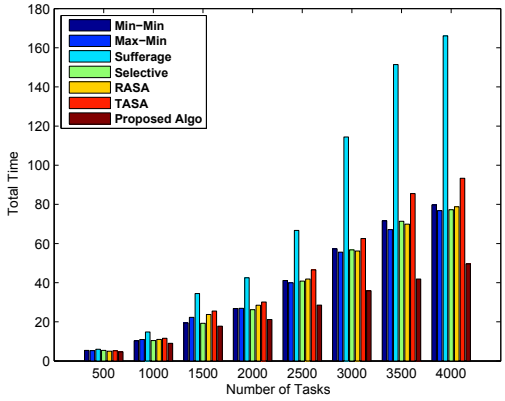

Fig. 4. Comparison of total time of the system for different algorithms for small sized tasks.

TABLE IV. COMPARISON OF DIFFERENT ALGORITHMS WHEN TASK SIZE=1000-2000; no_Vm = 100; $Vm\_Capacity$ = 1000MIPS- 2000 MIPS

No. of tasks		Min-Min	Max-Min	Sufferage	Selective	RASA	TASA	Proposed
500	MS	5.0	4.9	4.6	5.0	4.61	4.7	4.61
	AT	0.4	0.4	1.4	0.4	0.4	0.5	0.03
	TT	5.4	5.3	6	5.4	5	5.2	4.64
1000	MS	8.8	9.45	9.2	8.8	9.5	9.4	8.9
	AT	1.5	1.5	5.6	1.6	1.5	2.2	0.2
	TT	10.3	10.55	14.8	10.4	11	11.6	9.1
1500	MS	14.96	17.95	18.4	14.96	19.36	18.8	17.17
	AT	4.45	4.32	16.02	4.31	4.39	6.72	0.61
	TT	19.41	22.27	34.42	19.27	23.75	25.52	17.78
2000	MS	20.2	19.9	20.1	19.9	22	21.4	20.3
	AT	6.6	7	22.4	6.3	6.5	8.7	0.9
	TT	26.8	26.9	42.5	26.2	28.5	30.1	21.2
2500	MS	27.25	27.03	27.51	27.03	27.13	27.02	27.02
	AT	13.86	12.92	49.2	13.78	14.72	19.5	1.5
	TT	41.12	39.95	66.71	40.81	41.85	46.52	28.52
3000	MS	33.71	33.47	33.71	33.41	33.32	33.32	33.34
	AT	23.56	22.14	80.73	23.34	22.74	29.2	2.52
	TT	57.27	55.61	114.44	56.75	56.16	62.52	35.86
3500	MS	38.61	38.57	38.57	38.57	38.53	38.42	38.43
	AT	32.98	29.51	112.81	32.79	31.36	47.1	3.42
	TT	71.59	67.08	151.38	71.36	69.89	85.52	41.85
4000	MS	43.43	43.1	44.12	43.1	43.36	44.15	43.49
	AT	36.28	33.71	121.96	35.17	35.4	49.2	6.1
	TT	79.71	76.81	166.08	77.27	78.76	93.35	49.59

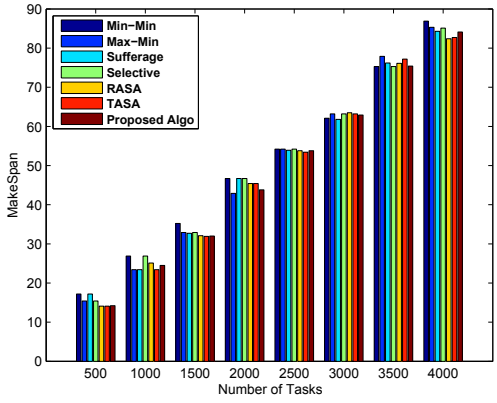

Fig. 5. Comparison of makespan time of the system for different algorithms for large sized tasks.

algorithms (in the last column) with other existing algorithms. Fig. 2 to 4 show the comparison graphs of the proposed algorithm with the existing competing six algorithms for makespan, allocation time (response time) and total time of the system for scenario 1, respectively. Fig. 5 to 7 show the comparison graphs for makespan, allocation time and total time

TABLE V. COMPARISON OF DIFFERENT ALGORITHMS WHEN $Task\_Size = 10000 - 20000$; $no\_Vm = 500$; $Vm\_Capacity$ = 1000 MIPS- 2000 MIPS

No. of tasks		Min-Min	Max-Min	Sufferage	Selective	RASA	TASA	Proposed
500	MS	17.2	15.4	17.2	17.2	14.1	14.1	14.2
	AT	1	1	9.3	1.1	1.1	2.7	0.4
	TT	18.2	16.4	26.5	18.3	15.2	16.8	14.6
1000	MS	26.9	23.4	23.4	26.9	25.1	23.4	24.5
	AT	4.7	4.9	30.4	4.9	4.9	10.1	1.1
	TT	31.6	28.3	53.8	31.7	30	33.5	25.6
1500	MS	35.2	32.9	32.7	32.9	32.1	31.9	32
	AT	10.2	10.4	89.4	10.3	10.5	25.9	3.1
	TT	45.4	43.3	112.1	43.2	42.6	57.8	35.1
2000	MS	46.7	42.9	46.7	46.7	45.4	45.4	43.8
	AT	19.7	19.8	179.9	19.9	19.7	54.7	5.9
	TT	66.4	62.7	226.6	66.6	65.1	100.1	49.7
2500	MS	54.2	54.2	53.91	54.2	53.8	53.4	53.8
	AT	35.1	35.3	250.1	35.4	35.7	87.4	10.2
	TT	89.3	89.5	254.01	89.6	89.5	141.8	64
3000	MS	62.1	63.2	61.8	63.2	63.5	63.2	62.9
	AT	51.9	51.7	560.2	52.4	52.9	143.4	15.4
	TT	114	114.9	622	115.6	116.4	206.5	78.3
3500	MS	75.3	77.9	76.2	75.3	76.1	77.2	75.4
	AT	69.4	69.2	690.7	69.7	69.8	192.7	20.72
	TT	144.7	147.1	766.9	145	145.9	269.9	96.1
4000	MS	86.9	85.3	84.3	85.1	82.4	82.7	84.1
	AT	89.1	89.3	820.5	89.7	89.9	240.8	27
	TT	176	174.6	904.8	174.8	172.3	323.5	111.1

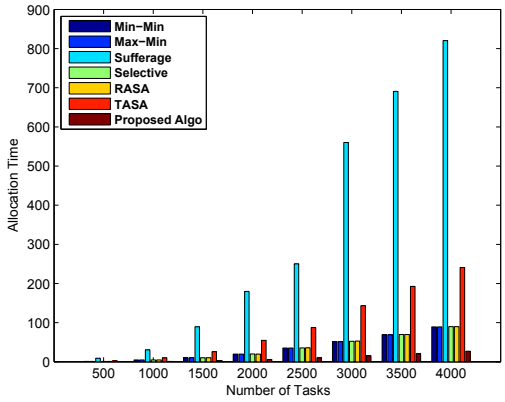

Fig. 6. Comparison of allocation time of the system for different algorithms for large sized tasks.

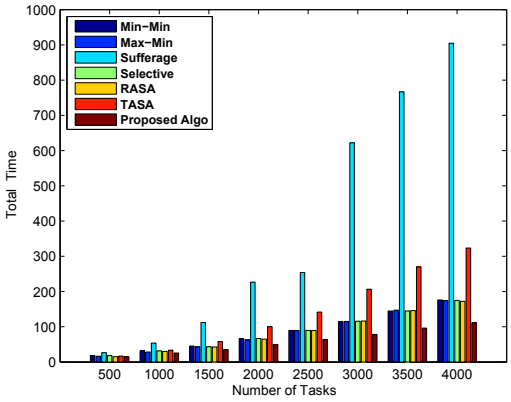

Fig. 7. Comparison of total time of the system for different algorithms for large sized tasks.

for scenario 2, respectively.

VI. CONCLUSION

In this paper, we have studied various task scheduling approaches in homogeneous and heterogeneous cloud environments. Moreover, we have proposed a heuristic algorithm by considering both larger and smaller size task simultaneously in heterogeneous cloud computing environment. Our proposed method considered a system model that consists of host model, VM model, and task model. We have used the ETC [19] to implement the proposed algorithm. Our proposed algorithm shows better efficiency with a large number of tasks in comparison with existing approaches. Therefore, we conclude that our algorithm is suitable for the execution of Big Data tasks in the cloud environment. In the future, we are planning to consolidate the real-time tasks with the partial modification of the system model and then verify the algorithm with the generated traffic.

REFERENCES

[1] Shawish, A., and Salama, M. "Cloud computing: paradigms and technologies." In Inter-cooperative collective intelligence: Techniques and applications, Springer Berlin Heidelberg, pp. 39-67, (2014).

[2] Younge, A. J., Von Laszewski, G., Wang, L., Lopez-Alarcon, S., and Carithers, W. "Efficient resource management for cloud computing environments." In IEEE International Conference on Green Computing, pp. 357-364, (2010, August).

[3] Assuncao, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., and Buyya, R. "Big Data computing and clouds: Trends and future directions." Journal of Parallel and Distributed Computing, 79, pp. 3-15, (2015).

[4] Fernandez, A., del Rio, S., Lpez, V., Bawakid, A., del Jesus, M. J., Bentez, J. M., and Herrera, F. "Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 4(5), pp. 380-409, (2014).

[5] Ma, T., Chu, Y., Zhao, L., and Ankhbayar, O. "Resource allocation and scheduling in cloud computing: policy and algorithm." IETE Technical review, 31(1), pp. 4-16, (2014).

[6] Kaur, R., and Kinger, S. "Analysis of job scheduling algorithms in cloud computing." International Journal of Computer Trends and Technology (IJCTT), 9(7), pp. 379-386, (2014).

[7] Nandgaonkar, S. V., and Raut, A. B. "A comprehensive study on cloud computing." International Journal of Computer Science and Mobile Computing, 3(4), pp. 733-738, (2014).

[8] Parsa, S., and Entezari-Maleki, R. "RASA: A new task scheduling algorithm in grid environment." World Applied sciences journal, 7(Special issue of Computer & IT), pp. 152-160, (2009).

[9] Taherian Dehkordi, S., and Khatibi Bardsiri, V. "TASA: A New Task Scheduling Algorithm in Cloud Computing." Journal of Advances in Computer Engineering and Technology, 1(4), pp. 25-32, (2015).

[10] Song, S., Hwang, K., and Kwok, Y. K. "Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling." IEEE Transactions on Computers, 55(6), pp. 703-719, (2006).

[11] Tripathy, L., and Patra, R. R. "Scheduling in cloud computing." International Journal on Cloud Computing: Services and Architecture (IJCCSA), 4(5), pp. 21-27, (2014).

[12] Tsai, C. W., and Rodrigues, J. J. "Metaheuristic scheduling for cloud: A survey." IEEE Systems Journal, 8(1), pp. 279-291, (2014).

[13] Maier, M., and Rimal, B.P. "Workflow Scheduling in Multi-Tenant Cloud Computing Environments." IEEE Trans. Parallel Distrib. Syst., 28, pp. 290-304, (2017).

[14] Bi, J., Li, B.H., Tan, W., and Yuan, H. "Temporal Task Scheduling With Constrained Service Delay for Profit Maximization in Hybrid Clouds." IEEE Trans. Automation Science and Engineering, 14, pp. 337-348, (2017).

[15] Nallakumar, R., Sengottaiyan, N., and Nithya, S. "A Survey of Task Scheduling Methods in Cloud Computing." International Journal of Computer Sciences and Engineering, 2, pp. 9-13, (2014).

[16] Katyal, M., and Mishra, A. "Application of selective algorithm for effective resource provisioning in cloud computing environment." arXiv preprint arXiv:1403.2914, (2014).

[17] Somasundaram, T.S., Govindarajan, K. "CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud." Future Generation Computer Systems, 34, pp. 47-65, (2014).

[18] Panda, S. K., Jana, P. K. "An Efficient Task Consolidation Algorithm for Cloud Computing Systems." In Distributed Computing and Internet Technology, Springer, pp. 61-74, (2016).

[19] Ali, S., Siegel, H. J., Maheswaran, M., and Hensgen, D. "Task execution time modeling for heterogeneous computing systems." In 9th IEEE Heterogeneous Computing Workshop, (HCW 2000), pp. 185-199, 2000.

[20] Puthal, D., Sahoo B., Mishra S., and Swain S. "Cloud computing features, issues, and challenges: a big picture." In International Conference on Computational Intelligence and Networks (CINE), 2015, pp. 116-123, 2015.

[21] Puthal, D., Wu, X., Nepal, S., Ranjan, R. and Chen, J. "SEEN: A Selective Encryption Method to Ensure Confidentiality for Big Sensing Data Streams." IEEE Transactions on Big Data (In press), 2017.

[22] Puthal, D., Nepal, S., Ranjan, R. and Chen, J. "A Synchronized Shared Key Generation Method for Maintaining End-to-End Security of Big Data Streams." In 50th Hawaii International Conference on System Sciences, pp. 6011-6020, 2017.

[23] Sahoo, S., Mishra, S., Sahoo, B., Puthal, D., and Obaidat, M. S. "Deadline-constraint services in cloud with heterogeneous servers." In International Conference on Computer, Information and Telecommunication Systems (CITS), pp. 20-24, 2017.

[24] Mishra, S., Khan, M., Sahoo, B., Puthal, D., Obaidat, M. S. and Hsiao, K. F. "Time efficient dynamic threshold-based load balancing technique for Cloud Computing." In International Conference on Computer, Information and Telecommunication Systems (CITS), pp. 161-165, 2017.