Journal of Information
2015 Vol.1, No.1, pp.36-52
ISSN(e): 2520-7652
DOI: 10.18488/journal.104/2015.1.1/104.1.36.52
© 2015 Conscientia Beam. All Rights Reserved.

REAL-TIME WORKLOAD SCHEDULING (RTWS) ALGORITHM FOR CLOUD

Sabout Nagaraju[1]† --- **Latha Parthiban**[2]

[1]*Assistant Professor, Pondicherry University; G.Pulla Reddy Engineering College, India*

[2] *PhD, Computer Science and Engineering, Pondicherry University, India*

ABSTRACT

Cloud computing is the revenue gain and most advanced technology that has tremendous advantages over other technologies. It can be used as a utility for executing large size of real-time programs. These programs are decomposed into multiple inter-dependent tasks and executed on the multiple virtual processors where the open research issue is to be minimized make-span of the scheduling tasks. Our research aims to address this issue and degenerate the schedule length approximately equal to the available number of virtual processors. We proposed a real-time workload scheduling algorithm that does very well in reducing the number of initial clusters. The experimental results show that the execution times for various kinds of the DAGs can be reduced as much as possible and improves the performance of the early load scheduling algorithms for distributed cloud environment.

Keywords: Clusters, Directed acyclic graphs, Distributed cloud, Virtual processors, Virtual machine.

1. INTRODUCTION

The Information and Communication Technology (ICT) stakeholders are migrating from in-house information technology solutions to the cloud computing solutions to avoid high cost of initial investments, training new personals, and operational costs. ICT stakeholders can improve their productivity more quickly and profitably. Cloud computing is a new computing and storage paradigm that provides larger amount data storage and computing power at nominal cost. Scheduling real-time workload in cloud computing is an open research problem to be addressed. Cloud users can gain their revenue from cloud systems, when the workload make-span is reduced as much as possible. In this research we have taken this issue and have presented an efficient solution to minimize the make-span of final clusters as much as possible where, we consider the real-time workloads given in [1-3] and used program [4] for generating task graphs.

Tasks in directed acyclic graph (DAG) are grouped as clusters based on their dependencies and executed on the multiple *virtual processors* (VP) available in the given virtual machines. In [5-9] various mechanisms are presented for tasks scheduling in cloud environment and minimized

† Corresponding author

make-span and monetary costs. The distributed cloud environment is still young and very few literatures are available. Generating workload schedule plan for the distributed systems applications is different from the ordinary scheduling. The distributed systems workload scheduling algorithm aims to achieve fairness between tasks, expected throughput from each and every individual task, and avoidance of the long waiting time and deadlocks. To the workload scheduling algorithms in cloud, these goals are mandatory and need to be minimized make-span and monetary costs. Several workload scheduling algorithms are developed over the last four decades to generate schedule plan to the distributed environments. In which, one of the leading tasks scheduling algorithm is the Task Duplication Based Scheduling (TDBS) algorithm [10] for heterogeneous systems.

The main goal of this research is to improve the performance of early tasks scheduling algorithms [10, 11] in cloud environment. The proposed RTWS algorithm merges the initial clusters efficiently, when the virtual processors available is less than required number of virtual processors. RTWS algorithm distributes the scheduled workload approximately equal to the available virtual processors on the distributed cloud.

This paper is further divided into four sections. Section 2, "Related Work", addresses the various early task-duplication based scheduling algorithms. We explained problem formulation in Section 3. Section 4, "Proposed Algorithm", presents generation and degeneration of the final clusters to the available number of virtual processors. Section 5, "Experimental results", explores the work done and compares with early discoveries and Section 6, "Conclusion", summarises the proposed research.

2. RELATED WORK

Since from 1975, large amount of researchers are contributing their valuable findings towards improving efficiency in workload scheduling algorithms for the distributed systems. Recently one of the fast emerging storage and computing paradigm is cloud distributed systems. In which, the open problem need to address is finding an efficient real-time workload scheduling algorithms. As part of this problem we proposed RTWS algorithm for cloud distributed systems. We have done literature survey on traditional and cloud workload scheduling algorithms. First we present the traditional related works and then we report cloud related works. In Rashmi and Dharma Agrawal [12] the authors Rashmi Bajaj et al. improved the performance of the task-duplication based scheduling algorithm [10] for heterogeneous systems. In this proposed research work, the scalability of the initial clusters to the less number of available virtual processors is not done effectively. We have described the shortcoming nature of the Rashmi Bajaj et al. proposed solution in Section III "problem formulation". We extended Rashmi Bajaj et al. approach in cloud environment.

In Hung-Chang, et al. [13] a fully distributed load rebalancing algorithm for distributed file systems in cloud is presented to solve the load imbalance issues. Where, the load is considered as a set of nodes and each node consist a set of files. Whenever insert or delete operations are performed on the files, the load imbalance will occur. In this research, the load rebalanced can be done using the system global knowledge in terms of less movement cost and high convergence rate. The authors suggested future direction that their proposed work can be extended to the cluster environment. This extension we have considered in our proposed work. In Sen [11] a Cost-Efficient Task Scheduling (CETS) algorithm for executing larger programs in the cloud is presented using two heuristic strategies for minimizing make-span and monetary cost. In this article, the concept of pare-to dominance is used to reduce execution time of the larger programs. Where, the scheduling plan to the virtual machines are generated based on the slack time of the *critical paths* (CP). The main short-coming of this research is that authors do not consider the case that the available number of processing elements in a virtual machine is less than the number of processing elements required by initial clusters. Our discovery is attempted to over-come the short-comings of the above related works. Our proposed RTWS algorithm equally distributes scheduling workload to the processing elements available in a VM or multiple virtual machines and it reduces the execution time as much as possible and also improves the performance of the above discussed scheduling algorithms in terms of make-span and monetary costs. In the following sections we prove that our investigation can be adopted in cloud environment and effectively balances the static workload.

3. PROBLEM FORMULATION

In our proposed algorithm we used the same terminologies given in Microsoft Dryad [14] and SDBS [10] are described below.

A. Computation of the Est and Ect

Assume:

τ: Set of node computation costs.

c: Set of non-zero edge communication costs.

The SDBS [10] computes the earliest start time (*est*) and earliest completion time (*ect*) for each node of the task dependency graph shown in the Fig.1. The earliest start time of a node is calculated as follows: Let *PRED(j)* be the set of predecessors to node *j* and c_{ij} be the communication cost between nodes *i* and *j*. Let *k* be the bottleneck node for *i*, such that max[$ect(k) + c_{ki}$ | k€*PRED(i)*]. Then, $est(i) = \max[ect(j) + c_{ji}$ | j€*PRED(i)*, j≠k, *ect(k)*] and the earliest completion time is simply the sum of est and the computational cost of the task and are shown in Table1. The level of each node in DAG can be computed as level (i) = max [sum [τ_i+ predecessors computation costs]].

B. Computation of Last and Lact

Compute the latest allowable start time (*last*) and latest allowable completion time (*lact*) by using fallowing expressions and are shown in the Table 1. For each task *i*, a favorite predecessor *fpre(i)* is assigned, which signifies that assigning both the task and its favorite predecessor will result in a lower parallel time.

$fpred(i) = \max_{j \in PRED(i)}(ect(j)+c_{j,i})$

$lact(i) = ect(i)$ for the exit node i

$lact(i) = \min(\min_{j \in SUCC(i), i \neq fpred(j)}(last(j)-c_{i,j}), \min_{j \in SUCC(i), i = fpred(j)}(last(j)))$

$last(i) = lact(i)-\tau(i)$

$fpred(i) = \max_{j \in PRED(i)}(ect(j)+c_{j,i})$

C. Cluster Generation

Traverses the above task dependency graph given in Fig. 1 in a reverse depth fist order and divides the tasks into clusters. Each cluster represents a path from the first unassigned task to the entry node and the tasks which are clustered together will execute on the same processor by using fallowing clustering Algorithm.

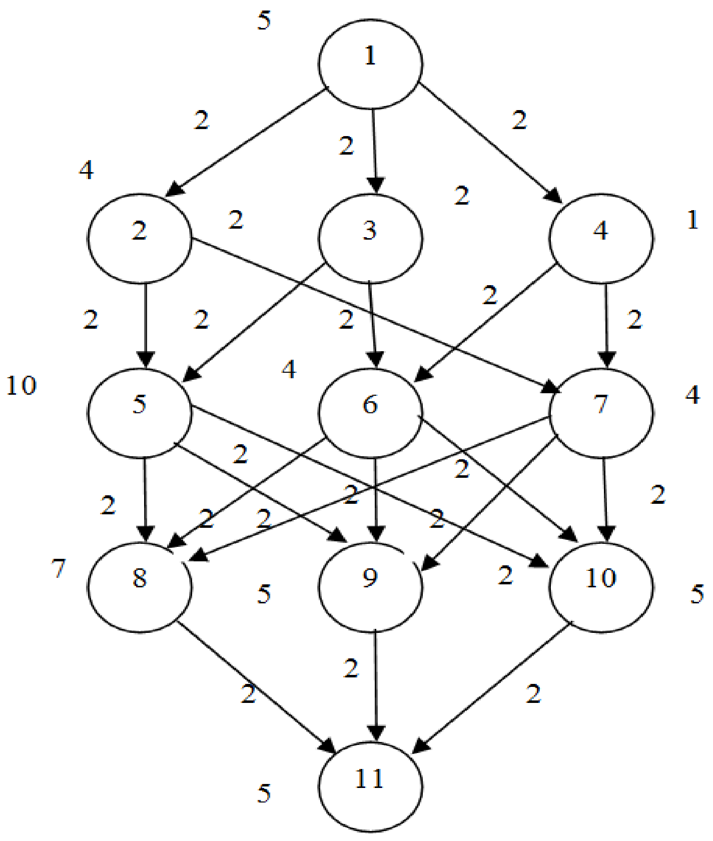

Fig-1. Task Dependency Graph

Table-1. Start and Completion Times

Node	Level	*est*	*ect*	*last*	*lact*	*fpred*
1	31	0	5	0	5	-
2	26	5	9	5	9	1
3	24	5	7	5	7	1
4	17	5	6	10	11	1
5	22	9	19	9	19	2
6	16	8	12	13	17	3
7	16	9	13	13	17	2
8	12	19	26	19	26	5
9	10	19	24	19	24	5
10	10	19	24	19	24	5
11	5	26	31	26	31	8

Algorithm-1.THESDBS Algorithm for generating clusters

```
Input:  DAG(v,e,$\tau$ ,c$i,j$)
pred(i): Set of predecessor tasks for task i.
SUCC(i):Set of successor tasks for task i.
QUEUE:Set of all tasks stored in ascendingorder of level.
Output: Task Clusters
Begin
$x$ = fist element of queue
Assign $x$ to an empty processor.
while(not all tasks are assigned to a processor)
{$y = fpred(x)$
if(($last(x)$ - $lact(y)$) $\geq$ $cx,y$ )then
if ($y$ has already been assigned to another processor)
$m$ = another predecessor z of $x$ (which has not yet been assigned)
else$m = y$
endif
else
if$y$ has already been assigned to another processor
for another predecessor $z$ of $x$ task $z$ has not yet been assigned to any processor
then  $m$ = z
else$m$=  $y$
endif
endif
assign$m$ to the current processor
$x = m$
if$x$ is entry  node
assign$x$ to the current processor.
$x$ = the next element in queue which has not yet been assigned to a processor
assign$x$ to a new processor and start the next cluster
endif
}
End
```

For Fig. 1, Algorithm 1 generates the initial clusters as shown in the Fig. 2.

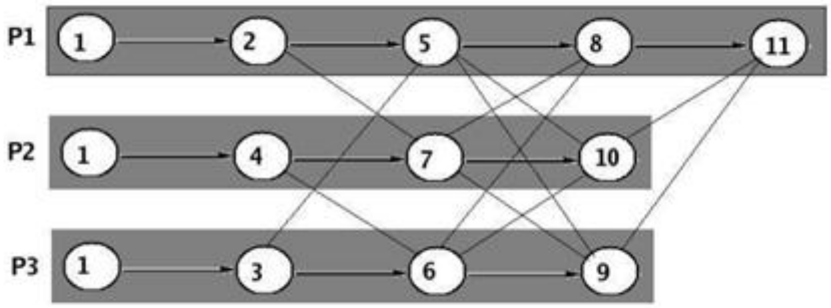

Fig-2. Initial Clusters

D. Scales the Number of Processors Required by the Initial Clusters

Scales the schedule based on following two cases:

Case 1: The available number of processors is higher than the number of processors required by the initial clusters: the SDBS [10] algorithm scales the schedule appropriately in an effort to obtain a lower parallel time by utilizing the extra or idle processors. For example, consider the scenario given in Fig. 2. The number of processors required is 3. In this there are two places where a favorite predecessor was not used when assigning tasks to processors. This is for tasks 10 and 9 for processors 2 and 3 respectively and for both tasks 9 and 10, task 5 is the favorite predecessor. Suppose the system have four processors available for the execution of this application, and then this algorithm assigns the tasks to processors as shown in Fig. 3.

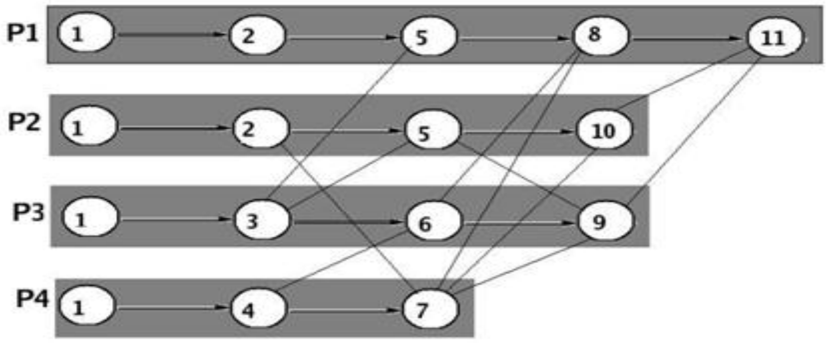

Fig-3. Task Allocation for four processors

Suppose the system have five processors available for the execution of this application, and then this algorithm allocates the tasks to processors as shown in Fig. 4.

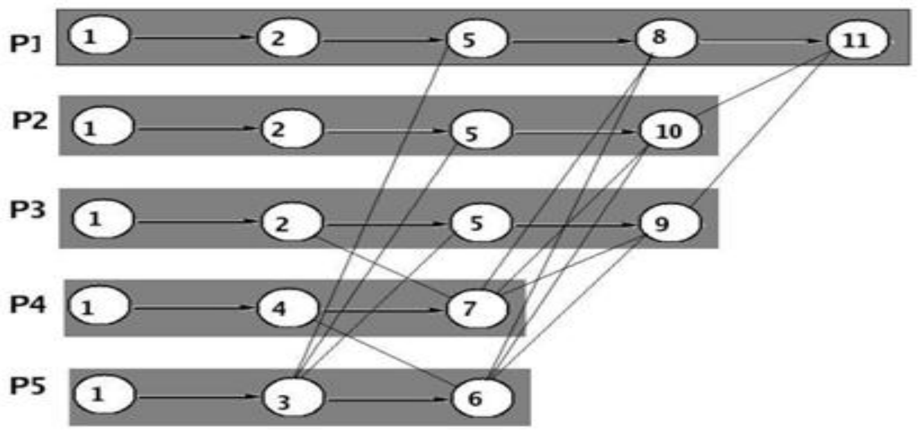

Fig-4. Task Allocation for five processors

Case2: The available number of processors are less than the number of processors required by the initial clusters: the SDBS [12]; [10] algorithm merges the task lists of different clusters. The number of processors required by the initial clusters is 3 as shown in above Fig. 1. If the available number of processors is less than 3 then the algorithm in Nitin and Dharma Agrawal [10] merges the task lists of different processors. For example, suppose the number of processors available is 2. The first step in reducing the number of processors is to find the values of *exec(i)* and *hole(i)* for each processor *i* and also the value of *maxexec*. The *exec(i)* of processors 1 would be equal to ($\tau(1) + \tau(2) + \tau(5) + \tau(8) + \tau(11)$) is 31, 2, and 3 are 15, and 16 respectively. Thus, *maxexec* is maximum execution time among all *exec(i)* is 31 and the *hole(i)= maxexec- exec(i)* for processors 1, 2, and 3 are 0, 16, and 15 respectively. This algorithm merges the task lists of processors (Pl,P2) or (P1,P3) and new task allocation list as shown in the Fig.5. Here, the *exec (P1)* is 46 and the *exec(P2)* is 16.

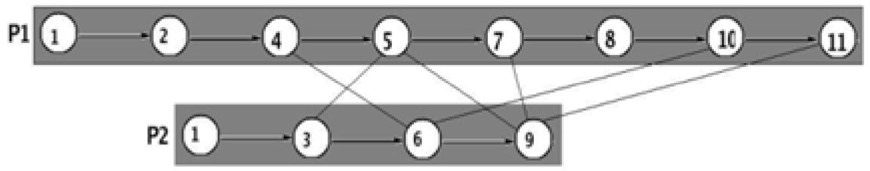

Fig-5. SDBS Final clustering processors

4. PROPOSED REAL TIME WORKLOAD SCHEDULING ALGORITHM

The proposed Real-Time Workload Scheduling Algorithm (RTWS) follows same assumptions as like algorithms in [10, 11] about the system and characteristics of the workload. The main assumption we have followed here is that the virtual machines can be heterogeneous, but the virtual processors in each virtual machine should be homogeneous. So that the computation and communication costs of the same task will be fixed between any two processors.

A. Description of the RTWS Scheme

In our scheme, the execution time of each task of a given DAG computed using Algorithm 2. Initial schedules generation and scaling performed using clustering algorithms given in Nitin and Dharma Agrawal [10]. If the initial schedules required more number of processors than the virtual processor available in a virtual machine, then the proposed algorithm works as follows:

Assume the number of initial schedules as n and the virtual processors available in a distributed cloud virtual machine as m. Step one computes execution times (*exec*) of initial clusters. Step two arranges *exec* in ascending order. Step three calculates the compactable clusters and non-compactable clusters as

- *compactable*= (n-m) indicates number of clusters to be merged and
- *non-compactable* = ($2m$-n) indicates number of clusters need not to be merged

Step four merges the clusters more efficiently by using the following steps repeatedly.

If n is greater than m, then

1. ($2m$-n) *is zero*: take sorted clusters and start merging from the middle, two clusters at a time and so on.
2. ($2m$-n) *is positive*: take ordered clusters except ($2m$-n) from the end and start merging from the middle, two at a time and so on.
3. ($2m$-n) *is negative*: take *abs* (($2m$-n)*2) of ordered clusters from the beginning and start merging from the middle, two clusters at a time and so on.

Repeat the above three steps for resultant clusters, where the number of resultant clusters is considered as n.

n= the number of resultant clusters.

The above mentioned steps will be repeated till resultant final schedules equal to virtual processors available. Our scheme degenerates final schedule length approximately equal to available virtual processors and does very well in reducing number of initial clusters.

Algorithm 2: Proposed RTWS Algorithm

```
Input: G (N,E)the DAG task dependency graph
N: tasks
  T: communication edges
m: the set of virtual processors available in a virtual machine
n: the set of initial schedules
Output: Final schedule plans which can be executed on the virtual processorsBeginComputes
start and completion times of each task using the sections 3.1 and 3.2.Initial clusters will be
generated using Algorithm given in Nitin and Dharma Agrawal [10].
if(n<m) then
    Scales the initial clusters n for the idle processors same as algorithm given in Nitin and
Dharma Agrawal [10].
else
 {
```

```
for ($p$=1to $n$)
    {
for($t$=1 to $nt$) //$nt$→number of task in each cluster
        $exec (p)+=\tau (t)$
    }
for ($p$=1 to $n$)
    {       if ($exec(p)>exec(p+1)$)
            Swap $exec (p)$ and $exec(p+1)$
    }
    Calculate non-compactable= (2m-n)
while ($n>m$)
    {
if (non-compactable==0) then
Take sorted clusters and start merging from the middle, two clusters at time and so on.
else if (non-compactable >0)then
Take ordered clusters except (2m-n) from the end and start merging from the middle, two clusters at a time and so on.
else
Take abs ((2m-n)*2) of ordered clusters from the beginning and start merging from the middle, two clusters at a time and so on.
end if
n= the number of resultant clusters.
    }
}
End
```

B. Running Trace of the RTWS Scheme

First step computes the *ests*, *ects*, *lasts* and *lacts* of a given graph as illustrated in Table 1. Step two generates the initial clusters as shown in Fig. 2. It can be seen that the processors required for the initial schedules is 3. Suppose the virtual processor available is two, then step three is not required. Step four performs the compaction of initial schedules as follow:

For the schedules depicted in Fig. 2, calculates the *exec(P1)* as 31, *exec(P2)* as 15 and *exec(P3)* as 16 and sorts these execution timings in ascending order as 15, 16, and 31. Then finds the number of clusters to be merged as 2 (i.e. *P2* and *P3*) and the numbers of clusters need not to be merged as 1 (i.e. P1). Finally merges the task clusters (P2, P3) as P2 and the final schedules are depicted in Fig. 6 and the analysis is reported in Table 2 and Fig 7. Where, the *exec (P1)* is 31 and the *exec (P2)* is 31. So, the proposed algorithm degenerates equal schedule lengths to the available number of virtual processors on the distributed cloud virtual machines.From this analysis the TDBS algorithm given in Nitin and Dharma Agrawal [10] generates imbalanced final schedule plans and the proposed algorithm produces balanced schedules. This analysis is represented in Section III.

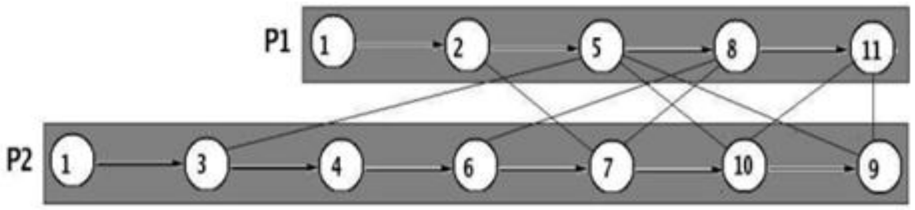

Fig-6. The proposed algorithm resultant schedules

Table-2. Final Schedule Execution Times with Two Virtual Processors

Virtual Processors	TDBS	Proposed Algorithm
	Execution Times	
P1	46	31
P2	16	31

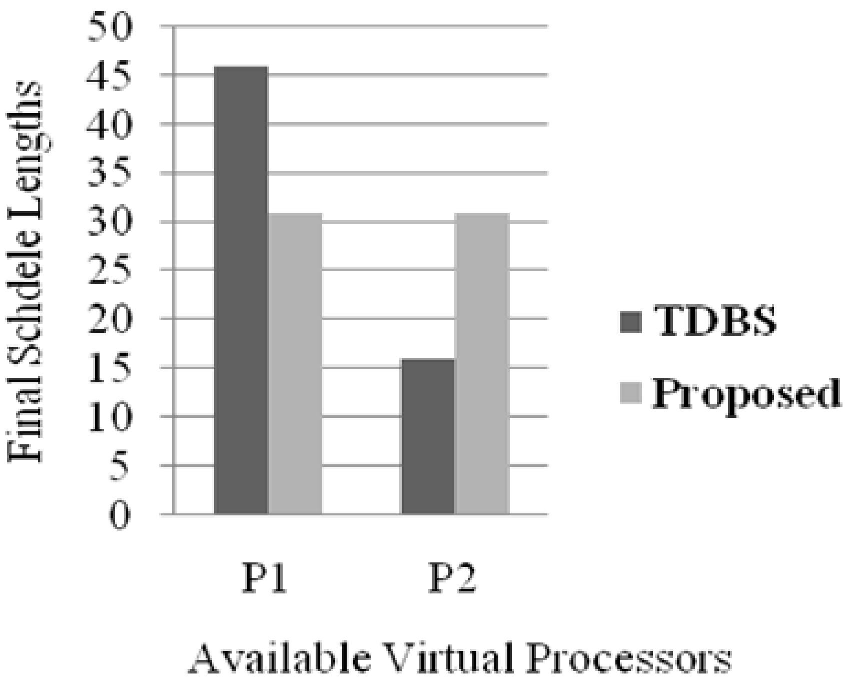

Fig-7. Resultant Schedules of the TDBS and RTWS schemes

C. Comparative Study

In this subsection we compares our RTWS algorithm with TDBS algorithm given in Nitin and Dharma Agrawal [10] for the DAG shown in Fig. 8. Where, we have considered the case that the virtual processors available is lesser than the processors required by the initial schedules. If the virtual processors available are two, then the schedule plans generated by the TDBS and RTWS algorithms are depicted in Fig. 9 and Fig. 10 respectively. And the final schedule plan execution timings are given in Table 3. In this case, the both the algorithms take same execution times for a given task dependency graph.+

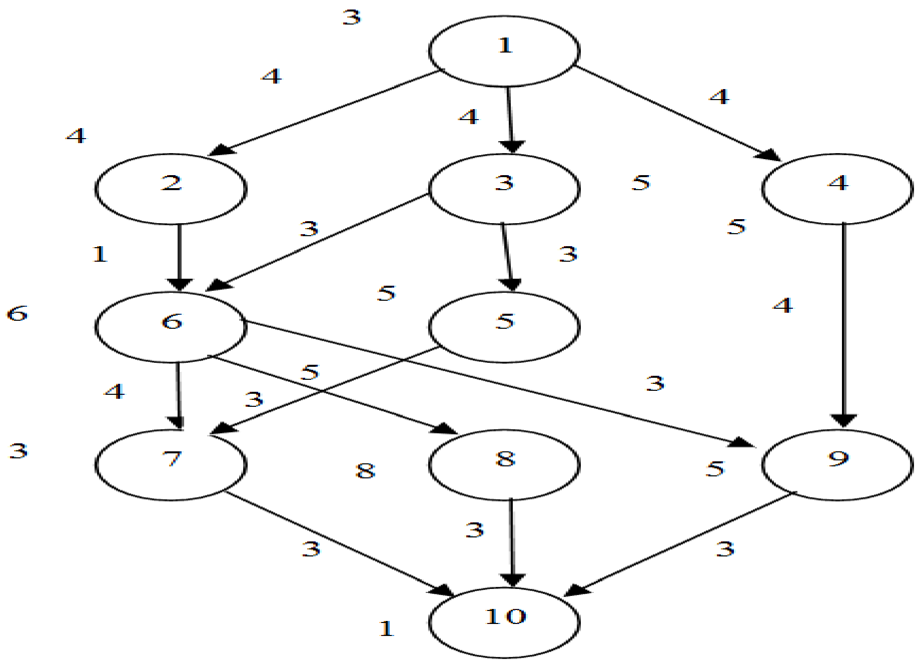

Fig-8. Task Dependency Graph

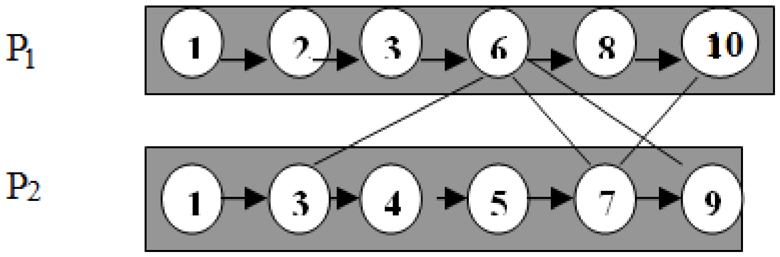

Fig-9.TDBS resultant clusters with two Processors

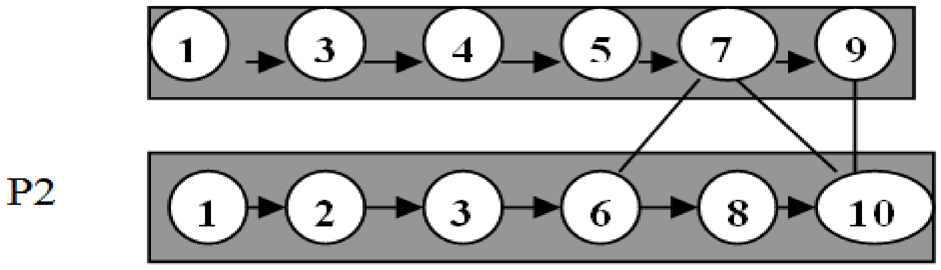

Fig-10. Proposed Algorithm Schedules with two Processors

Table-3. Final Schedule Execution Times With Two Processors

Virtual Processors	TDBS Algorithm	Proposed Algorithm
	Execution Times	
P1	27	27
P2	27	27

Suppose the virtual processors available are three, then the schedule plans generated by the algorithm [10] and the proposed algorithm are shown in Fig. 11 and Fig. 12 respectively. And the final schedule plan execution timings are reported in Table 4. In this case, the proposed algorithms generated efficient schedule plan than the algorithm in Nitin and Dharma Agrawal [10] for a task dependency graph given in Fig. 8.

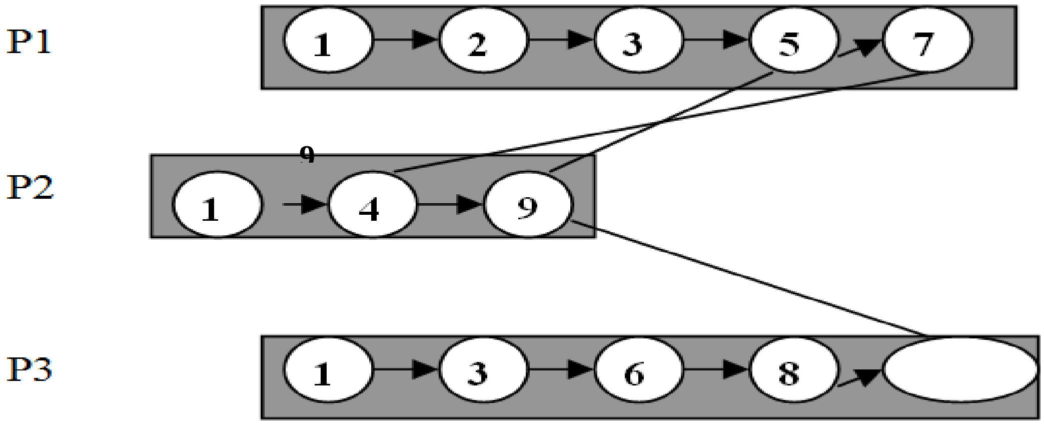

Fig-11.TDBS resultant clusters with three Processors

From the above study, the TDBS algorithm in Nitin and Dharma Agrawal [10] generates imbalanced final schedule plans and the proposed algorithm produces balanced schedules. This analysis is represented in Fig. 13.

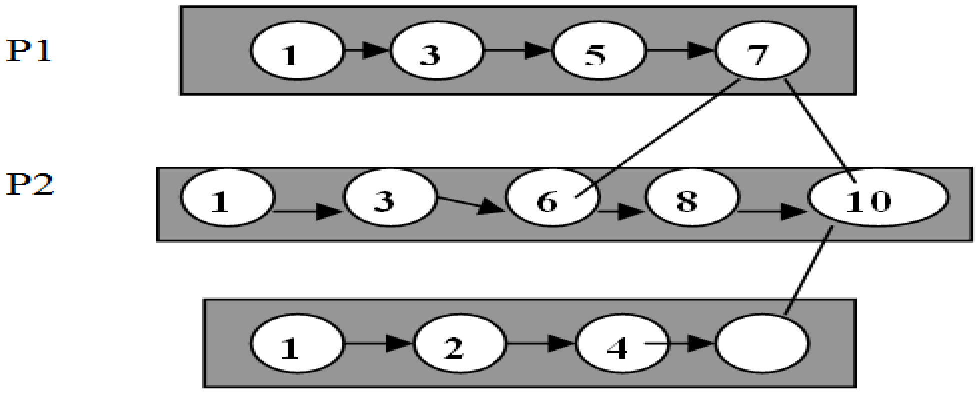

Fig-12. Proposed Algorithm clusters with three Processors

Table-4. Execution Timings of Three Processors

Virtual Processors	TDBS Algorithm	Proposed Algorithm
	Execution Timings	
P1	20	18
P2	16	16
P3	23	23

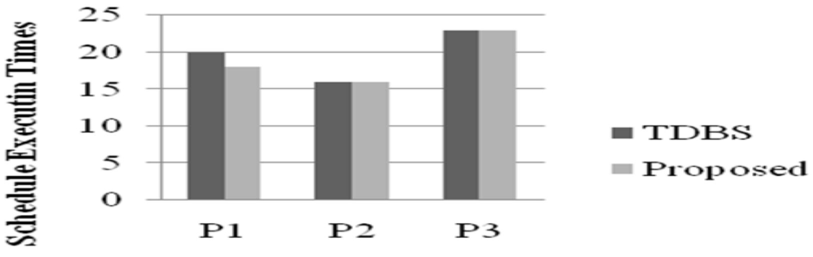

Fig-13. Schedule plan comparison of TDBS and Proposed algorithm

D. Comparison Metrics

i. Make-Span

The execution time required for the entire DAG is called *make-span.* That is sum of all the final clusters execution time and is defined as follow:

Make-span=Σ *exec (i)*, where *i is*1 *to m.*

*m*indicates the set of available virtual processors. So, the *make-span* taken by TDBS algorithm is 59 and our proposed algorithm is 57 for the DAG given in Fig. 8.

ii. Schedule Length Ratio

In our simulation, we mainly consider *make-span* as the comparison factor for various kinds of real-time task graphs given in [1-3]. Another key parameter we have taken is *Schedule Length Ratio* (*SLR*). It can be defined as follow:

$$SLR = \frac{\text{Make-span}}{\sum_{v_i \in CP} \min_{m_j \in M}\{t(i,j)}$$

Where, the denominator indicates that the sum of minimum computation times of the critical path tasks which can be executed on a virtual machine m_j. The critical path taken here reduces the cumulative computation times of the DAG tasks. So, the TDBS algorithm generated schedule length ratio is 3.6875(i.e., 59/16) and the *SLR* generated by the proposed algorithm is 3.5625 (i.e., 57/16) for the DAG given in Fig. 8.

iii. Scalability Factor

The third key factor we consider is *Scalability Factor* (*SF*) and it can be defined as a degree of the scalability in initial schedule lengths to the available virtual processors. If *Scalability Factor is* lesser than one, then the initial schedules need to be merged. Otherwise, the initial clusters are duplicated on idle virtual processors. For the DAG given in Fig. 8, the *SF* is 0.6 when the available number of virtual processor is 3.

5. EXPERIMENTAL RESULTS

Our proposed RTWS algorithm is used for generating and degenerating schedule plans. We have considered five different types of task graphs to compare and evaluate the effectiveness of the RTWS scheme with CETS and TDBS algorithms. And we have used larger number of task graphs generated randomly on the real-time workload given in [1-3] as inputs.

A. Experimental Setup

We have used Cloud Simulator described in Rodrigo Calheiros, et al. [15] for experimental setup of the proposed algorithm. It is a toolkit used for simulation and modeling of the distributed cloud environments. We have compared and tested the proposed algorithm with the algorithms given in [10, 11] for various kinds of DAGs that are generated from real-time workloads such as Nephele project [1] Pregel project [2] SPEC fppp and sparse matrix solver [3] and one randomly generated diamond graph. Input DAGs are generated using the program [4] Benchmark Standard Task Graph Set [3] and TGFF suite [16]. In the RTWS algorithm implementation, we considered the node computation and communication costs in milliseconds and calculated the make-span in seconds.

B. Performance Evaluation

We have observed the schedule lengths in form of make-span to the available number of virtual processors (VP). Table 5 and Fig. 14 describes the make-span cost taken by the CETS, TDBS and proposed algorithm for the task graphs with 1500, 3000, 4500, 6000 and 10000 tasks. And with required number of virtual processors for each task graph is 210, 300, 390, 450, and 500.

Table-5. Make-Span Produced By Algorithms

Tasks	VP	CETS	TDBS	RTWS
		Final Schedule Make-span		
1500	210	44	45	39
3000	300	65	63	51
4500	390	66	69	67
6000	450	83	79	72
10000	500	80	89	72

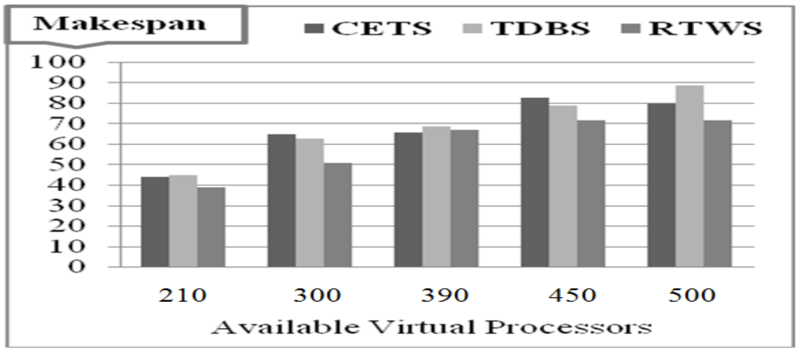

Fig-14. Performance comparison of CETS, TDBS and RTWS for the real-time task graphs

The very important key factor for reducing execution time of the task graphs is *SLR*. Table 6 and Fig. 15 shows the average schedule length ratios generated by CWTS, TDBS and RTWS algorithms.

Table-6. The Average Schedule Length Ratios

Tasks	VP	CETS	TDBS	RTWS
		Schedule Length Ratio		
1500	189	5.7805	5.7923	5.6539
3000	245	6.5367	6.2538	5.0750
4500	318	7.8225	7.8401	7.5946
6000	398	8.9764	8.7427	7.9863
10000	410	9.3486	9.7829	8.9705

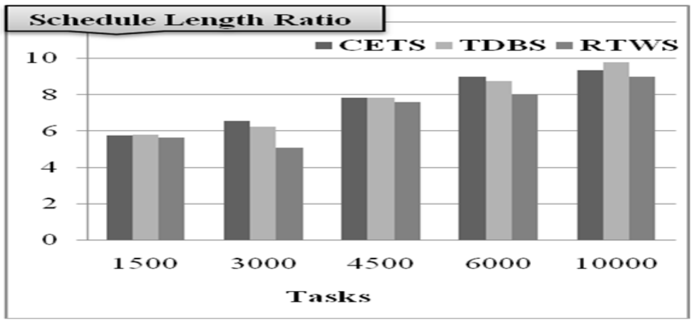

Fig-15. *SLR* comparison of CETS, TDBS and RTWS for the real-time task graphs

The effective factor we used for evaluating performance of the various task scheduling algorithms is *SF*. We consider the DAG with 1000 tasks, actual required number of virtual processors is 500 and available virtual processors are 110, 210, 310, 410, and 460. Fig.16 describes the make-spans generated by CETS, TDBS and RTWS algorithms with different scalability factors.

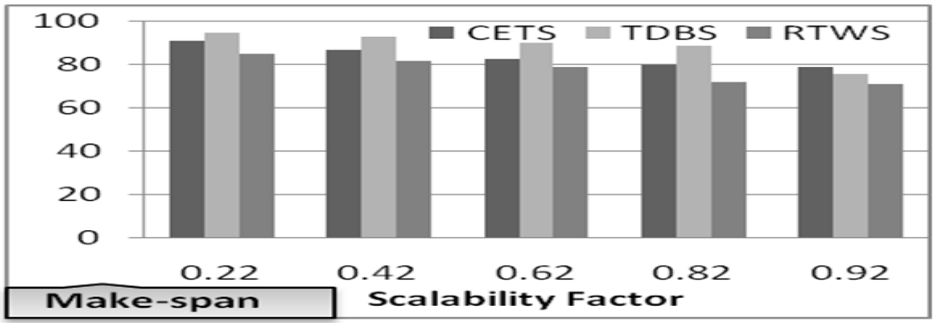

Fig-16. Performance comparison of CETS, TDBS and RTWS for the DAG with 1000 tasks

Our discovery concludes that proposed scheme reduces make-span cost and improves the overall performance of the CETS and SDBS algorithms.

6. CONCLUSION

The proposed real-time workload scheduling algorithm obtains maximum benefits for the cloud users from the distributed cloud systems by an efficient task scheduling. This algorithm reduces the number of initial clusters and degenerates balanced schedule plan to the available processing elements. The series of experimental results show that our proposed algorithm reduces the make-span cost of the real-time applications as much as possible and improves the run-time efficiency. In our research, we have considered the static workload and this work can be extended to the dynamic workload of the real-time applications in cloud. Further enhancement in our work can be done by using meta-heuristic techniques. In our future work we are also planning to address various security violations raises in real-time workflow for the inter-cloud data transfer.

REFERENCES

[1] D. Warneke and N. Kao, O., "Efficient parallel data processing in the cloud," in *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, ACM*, 2009.

[2] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski Pregel, "A system for large-scale graph processing," in *Proceedings of the 2010 International Conference on Management of Data, ACM*, 2010, pp. 135-146.

[3] The Standard Task Graph Set, Available: http://www.kasahara.elec.waseda.ac.jp/schedule, 2008.

[4] DAG Generation Program. Available http://www.loria.fr/suter/dags/html, 2010.

[5] W. Paul, "A multi-level security model for partitioning workflows over federated clouds," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 1, pp. 1-15, 2012.

[6] A. Amit and K. Padam, "Economical duplication based task scheduling for heterogeneous and homogeneous computing systems," presented at the WEE International Advance Computing Conference (LACC 2009) Patiala, India 6-7 March 2009, 2009.

[7] M. Karthikeya, P. Gajjala, and B. Dinesh, "Temporal partitioning and scheduling data flow graphs for reconfigurable computers," *IEEE Transactions on Computers*, vol. 48, pp. 579-590, 1999.

[8] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in google compute clusters," in *Proc. 2011 ACM Symposium on Cloud Computing*, n.d, pp. 1–3:14.

[9] L. Xiao Cheng, W. Chen, Z. Bing Bing, C. Junliang, Y. Ting, and Y. Albert Zomaya, "Priority-based consolidation of parallel workloads in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 1874-1883, 2013.

[10] A. Nitin and P. Dharma Agrawal, "Enhancing the schedulability of real-time heterogeneous networks of workstations (NOWs)," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 1586-1599, 2009.

[11] S. Sen, "Cost-efficient task scheduling for executing large programs in the cloud," *Parallel Computing*, vol. 39, pp. 177-188, 2013.

[12] B. Rashmi and P. Dharma Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, pp. 107-118, 2004.

[13] H. Hung-Chang, C. Hsueh-Yi, S. Haiying, and C. Yu-Chang, "Load rebalancing for distributed file systems in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 951-962, 2013.

[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad, distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Operating Systems Review*, vol. 41, pp. 59-72, 2007.

[15] N. Rodrigo Calheiros, R. Rajiv, B. Anton, A. F. C´esar De Rose, and B. Rajkumar, *CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*: Wiley Online Library. Available: wileyonlinelibrary.com [Accessed 24 August 2010], 2010.

[16] R. Dick, D. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. Sixth Int'l Workshop Hardware/Software Co-Design (CODES/CASHE '98)*, 1998.