Deadline-Constrained Task Scheduling with Load Balancing in Cloud Computing Environment

Harish Kumar Patnaik, Manas Ranjan Patra

Abstract:. Cloud computing environment is a conglomeration of computing resources that are maintained to provide a plethora of services to users as utility on a pay-per-use basis. When users request for services, a cloud provider has to allocate relevant computing resources not only to complete a task but also to satisfy the user requirements such as deadlines. This necessitates the design of suitable scheduling algorithm for effective utilization of available resources at the provider's end. While doing so one of the concerns of the provider is to balance the load uniformly across resources. In this paper, the Max-Min algorithm has been extended by incorporating the concept of free time in order to effectively utilize the cloud resources through load balancing and at the same time meet the deadlines of individual tasks.

Index Terms : Cloud Computing, Deadline, Free Time,. Load Balancing, Max-Min, Task Scheduling

I. INTRODUCTION

Cloud is a collection of storage unit, servers and services that exist over the internet where users have a convenient, on-demand access to a shared pool of resources. Users can send request over a network for a cloud service provider to respond. It is a model based on the concept of sharing of virtual resource as per user requirement. It works on two basic concepts; viz., abstraction and virtualization. A cloud user has no information as to where data are stored and in which physical machines the applications run. Virtualization enables creation of virtual machines (VMs) or replicas of computing resources which are provisioned on demand by the users and billed as per usage. The resources are scalable as per the requirement.

Cloud based applications suffer from the latency problem existing in the network. The current technology is based on live migration of virtual machines for proper resource allocation to jobs. But, the migrations add overhead to the network. The common approach to handle this problem is to develop some possible schedules and choose the most suitable one which would minimize migration and maximize resource utilization. Yet another requirement is that jobs are to be completed within user specified deadlines. Thus, the present study focuses on developing task scheduling strategies that can ensure completion of tasks within deadlines and optimal resource utilization through load balancing in order to minimize migration of tasks.

Revised Manuscript Received on July 05, 2019

Harish Kumar Patnaik, School of Computer Engineering, KIIT University, Bhubaneswar, India.

Manas Ranjan Patra, Department of Computer Science, Berhampur University, Berhampur, India.

II. MOTIVATION AND RELATED WORK

When users send request to be executed, the cloud provider refers to the service level agreement and passes it to the cloud task scheduling process. The process checks the status of available resources and assigns the tasks to different resources as per requirement. Many task scheduling algorithms have been proposed for efficient allocation of resources [7][8][10]. Completing all the tasks in time and at the same time utilize the resources optimally is a challenging task. Some of the approaches are as follows:

A. Advance Reservation (AR): Resource reservation is done in advance so that they are available at a specified time.

B. Best Effort: Request for resources are placed in a queue. Resource allocation is done based on availability.

C. Immediate: On receiving a resource request either the requested resources are allocated immediately or the request is rejected due to unavailability of resources.

Heuristic approaches for resource allocation have been proposed in [11] [12] that consider factors such as:

- ✓ Minimum Execution Time – The scheduler assigns tasks to the machines where the execution time is minimum. But, here the availability of a machine at the time of actual scheduling is not considered.
- ✓ Minimum Completion Time – The scheduler selects that machine in which the expected completion time of task is minimum among all the available machines. However it considers only the load of the machine without considering minimum_execution_time. Completion_time of a task is computed as the sum of time taken for execution on a machine and the availability time of that machine.

Some dynamic algorithms to generate the static resource allocation have been proposed in [14] such as:

- ✓ Cloud List scheduling – Here the resources are assigned to tasks on priority basis. The assigned task starts its execution only after all it's previous tasks have finished and the resource is free.

✓ Cloud Min-min scheduling –In this approach, first the task whose execution time is minimum on all resources is figured out and then it is allocated to that resource for which the completion _time is minimum. This procedure is followed for all the remaining tasks.

El-Sayed et al. [8] proposed Max-Min scheduling algorithm to allocate the tasks to the available resources which is slightly different from Min-min scheduling. In this approach, first the task with highest execution_time on all resources is found and then it is allocated to that resource for which the completion _time is minimum.

Motivational example

An attempt has been made to compare the performance of these two algorithms for the purpose of scheduling. Let us consider, three different resources and five different tasks. Table A presents the processing speed of the resources and Table B presents the size of all task. Considering these data, execution_time and expected completion_time of the tasks are calculated using Min-Min algorithm as shown in Table C. The struck out figures indicate that those resources are not allocated to the corresponding row task rather that task is assigned to the resource that is left out. Table C presents the resultant schedule according to Min-Min algorithm.

TABLE A. RESOURCE CAPACITY

Resources	Processing Speed(*MB/Sec*)
R1	10
R2	8
R3	5

TABLE B. TASK SPECIFICATION

Tasks	Task Size(*MB*)
T1	10
T2	15
T3	20
T4	25
T5	50

TABLE C. EXECUTION TIME(EXPECTED COMPLETION TIME) MIN-MIN ALGORITHM

Tasks\ Resources	R1	R2	R3
T1	1(1)	~~1.25(1.25)~~	~~2(2)~~
T2	~~1.5(2.5)~~	1.875(1.875)	~~3(3)~~
T3	2(3)	~~2.5(4.375)~~	~~4(4)~~
T4	~~2.5(5.5)~~	3.125(5)	~~5(5)~~
T5	5(8)	~~6.25(11.25)~~	~~10(10)~~

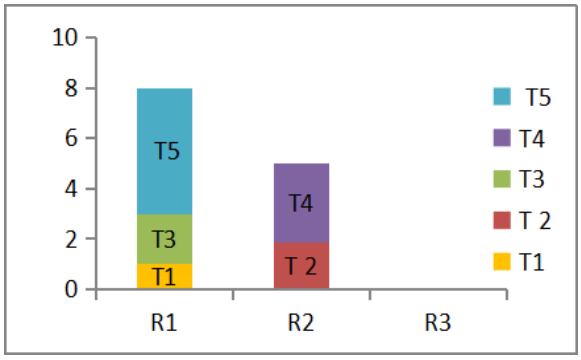

Fig 1. Gantt Chart using Min-Min algorithm

Fig 1 presents the Gantt chart for the schedule generated by the Min-Min algorithm. It shows the tasks T1, T3, T5 are allocated to resource R1 and the tasks T2, T4 are allocated to resource R2. It depicts the total makespan, which is calculated as Maximum of all RT_i, where RT_i is the ready_ time of each resource after exeution which is 8 seconds in this case.

TABLE D. EXECUTION TIME(EXPECTED COMPLETION TIME) MAX-MIN ALGORITHM

Tasks\ Resources	R1	R2	R3
T5	5(5)	~~6.25(6.25)~~	~~10(10)~~
T4	~~2.5(7.5)~~	3.125(3.125)	~~5(5)~~
T3	~~2(7)~~	~~2.5(5.625)~~	4(4)
T2	~~1.5(6.5)~~	1.875(5)	~~3(7)~~
T1	1(6)	~~1.25(6.25)~~	~~2(6)~~

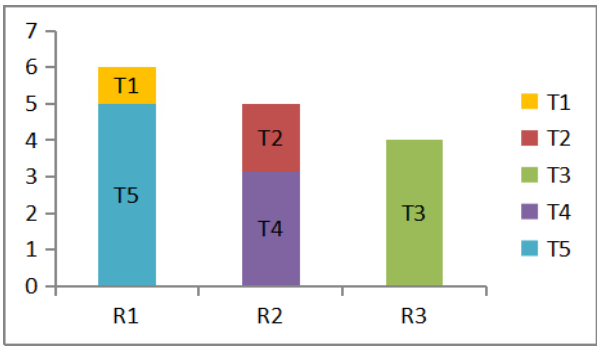

Fig 2. Gantt Chart using Max-Min algorithm

Again the execution_time and the expected completion_time are calculated for the same tasks and resources according to Max-Min algorithm as shown in Table D. Fig 2 presents the Gantt chart for the schedule generated using Max-Min algorithm. It shows that the tasks T1, T5 are allocated to resource R1, the tasks T2, T4 are allocated to resource R2, and the task T3 is allocated to resource R3. Now the total makespan in Fig 2 is 6 seconds and it is less than that of Min-Min algorithm.

Unlike Min-Min, where no task is allocated to the resource R3, here it is noted that the Max-Min algorithm allocates the tasks evenly across the resources to some extent. Also it's makespan is better than Min-Min scheduling. In this example, we show the improvement of Max-Min algorithm over Min-Min algorithm which motivated us to carry forward with Max-Min algorithm. Also it is observed that using Max-Min scheduling algorithm no such model has been formulated yet using free time concept along with task's deadlines and load balancing of resources. Therefore, it is believed that an efficient model can be designed with such an optimizing technique.

III. PROPOSED APPROACH

In this paper, we propose two algorithms - a scheduling algorithm and a load balancing algorithm for effective utilization of resources. It is an extension of Max-Min algorithm. Considering all the tasks with their respective deadlines, if any, a feasible schedule is prepared using Max-Min policy as given in Algorithm 1. Here, on every resources the Execution time(ET) of each task is computed using formula:

$$ET = \frac{Task\ Size(MB)}{Resource\ Capacity(\frac{MB}{Sec})}$$

Similarly, Completion time is calculated using the formula $CT_{ij} = ET_{ij} + RA_j$ [15] where RA represents the resource availability time. Since at the beginning all resources are free, the resource availability time is set to zero for all. Then the task of maximum size is considered to be allocated to a resource where the task completes in lowest time. The same process is followed for the next largest size task till all the tasks are considered.

Algorithm 1.
Arrange the tasks in list TL in decreasing order of their size.

1. ***for*** *each task* T_n *in TL,*
   for *each resource* R_m,
   Calculate $CT_{mn} = ET_{mn} + RA_m$
   end for
   end for
2. ***while*** *(the list TL is not empty)* ***do***
   i. *Consider the first task* T_f *from TL*
   ii. *Assign* T_f *to the resource* R_j *for which* CT_{fj} *is minimum*
   iii. *Delete* T_f *from TL*
   iv. *Recalculate* RA_m
   v. *Recalculate* CT_{mn} *for all* T_f

Now the schedule generated by Algorithm 1 is checked whether all the tasks can be completed before their respective deadlines. If all the tasks can meet their deadlines, then the schedule is considered as appropriate. Otherwise, the next algorithm Algorithm2 is applied to reschedule the tasks that have failed to meet their deadlines where the Free time concept is used. The time gap between the deadline and completion time is the free time of the resource. Often there are tasks that are completed before their respective deadlines. As per the service level agreement, all the jobs need to be completed by deadline. But by completing the jobs much before the deadline will not benefit the service provider by any means. Therefore, this free time of the resources can be utilised by executing those tasks which failed to meet their deadlines. The Free time is calculated for those tasks which have completed their execution before deadline by using the formula:

Free time(FT) =Deadline (DL) – Completion time (CT)

Algorithm 2.
Prepare two sets of tasks X and Y, where X is the set of task that met the deadline and Y for others.

1. ***for*** *each task* T_x *in X, assigned to a resource* R_j
   Calculate $FT_x = DL_x - CT_{xj}$
   end for
2. ***for*** *each resource* R_j
   for *each and every task* T_x *allocated to* R_j
   i. *prepare a matrix* FTM_{jx}
   ii. *Map* FTM_{jx} *with* FT_x *to indicate which* T_x *has how much* FT_x *on a resource* R_j
   end for
   end for
3. ***while*** *(each task* T_y *in Y is considered)* ***do***
   I. ***for*** *each resource* R_j
   if $ET_j(T_y)$ *on* R_j *is less than* FTM_{jx} *and also less than* DL_y
   i. *Consider the* FT_x *of the next upper task* T_x *assigned to* R_j *till (* $DL_x > CT_{xj}$ *and* $ET_j(T_y)$ *on* $R_j \leq FTM_{jx}$ *). Mark this location as* P_j.
   ii. *if* T_y *is allocated to* R_j *then calculate the balance computing capacity as* $Cleft_j$= *(FT of the last allocated task on* R_j - $ET_j(T_y)) \times Capacity(R_j)$
   end if
   end for
   II. *Assign* T_y *to that resource* R_j *which has minimum* $Cleft_j$
   III. *Recalculate* FTM_{jx} *from the location* P_j *to the last task assigned to* R_j *as* $FTM_{jx} = FTM_{jx} - ET_j(T_y)$
   IV. *Recalculate* CTM_{xj} *for* T_y

A matrix is formed to represent the Free time of each of the resources for each of the task assigned to them. Then the failed tasks are checked whether they can be accommodated within these Free times of all the resources so they can be completed before their deadline without affecting the schedule of other task. If the failed task can be accommodated in more than one resources, then it will be allocated to that resource where the left over computation is minimum because the leftover computation will not be able to satisfy any other task and the resources which have more leftover computation can satisfy new incoming tasks.

Algorithm 3.

1. *Find makespan (Sm) of all the resources.*
2. ***while*** *(highest makespan(Sm1) – 2nd highest makespan (Sm2) > 1)* ***do***
   - i. *Find the resource Rm with highest makespan Sm1*
   - ii. *Take the last task Ti on Rm*
   - iii. *Find the resource Rj which produces minimum_completion_ time CTmin if task Ti is allotted at end of Rj*
   - iv. *if CTmin < Sm*
     - *Reallocate Ti to resource Rj*
     - *Recalculate the ready_time of both Rm andRj*
     - *Recalculate Sm*

Then it is checked whether all the resources are evenly loaded or not. It can be done by comparing the makespan of all resources. If the makespan of the resources are not close to each other then the resources are not uniformly loaded. For this a load balancing scheme is proposed in Algorithm 3 which distributes the tasks among all resources.

IV. RESULT ANALYSIS

For analysis purpose, a set of resources and tasks are considered as shown in the tables E and F below.

TABLE E. TASK SPECIFICATION

Tasks	Task Size(MB)	Deadline(secs)
T1	15	8
T2	25	12
T3	34	23
T4	42	24
T5	21	21
T6	58	22
T7	80	25
T8	71	19
T9	22	20
T10	12	23

TABLE F. RESOURCE CONFIGURATION

Resources	Processing Speed(MB/sec)
R1	10
R2	8
R3	5

On applying Algorithm 1 to these inputs, the schedule obtained is as shown in Figure 3. In this schedule it may be noted that T1 and T2 will complete their execution in 16.2 sec and 14.7 sec respectively on R1 and will not meet their deadlines (8 secs and 12 secs respectively). Therefore, to accommodate these two tasks, Algorithm 2 is used. The free time is calculated for the remaining tasks and a matrix FTM is formed. The available free time on each of the resources and the allotted tasks are shown in Figure 4 and Figure 5.

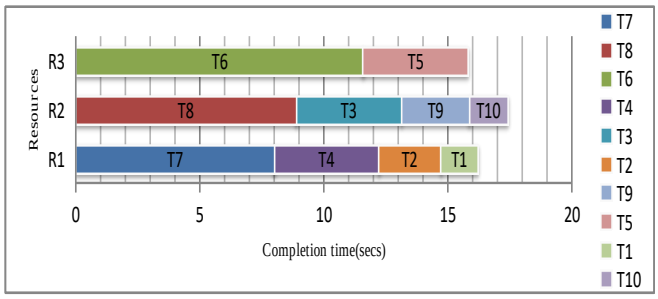

Fig 3. Generated schedule using Algorithm 1

R1	17 *secs*	11.8 *secs*		
R2	10.125 *secs*	9.875 *secs*	4.125 *secs*	5.625 *secs*
R3	10.4 *secs*	5.2 *secs*		

Fig 4. Free time matrix on each resource FTM

R1	*T7*	*T4*		
R2	*T8*	*T3*	*T9*	*T10*
R3	*T6*	*T5*		

Fig 5. Scheduled task on resources corresponding to FTM

Now the tasks which are not meeting their deadlines, are checked whether they can be reallocated to any resources. First the task T2 is considered as its size is larger than T1. Then execution time of T2 on R1 (2.5 secs) is checked whether it is lesser or equal to the free time of the last task T4 assigned to R1 i.e. 11.8 secs which is true. It is also verified if execution time of T2 on R1 is less than or equal to deadline of T2 which is also true. The reason behind this check is that, if the condition fails then even if T2 had been assigned first to R1, T2 can never be completed before deadline by R1. After verifying this, the FT of the immediate previous task of the current one is compared. This procedure is performed for every task previously allotted to R1 until T2 can be accommodated in FTs without affecting deadline of other tasks and this point is marked as P1. Similarly, points P2 and P3 are found for resources R2 and R3. But T2 will be allocated to that resource where the left over computation time is less.

If it is decided to allocate T2 to R1 then the computation left in R1 is calculated as per the formula in step 3.I.ii. as Cleft = (11.8-2.5)×10 which is 93 MB which means that with the leftover free time on R1, tasks of 93 MB can be further accommodated. Similar checks are performed with other resources R2 and R3 for assignment of T2. In R2 the location P_2 is the starting location and the computation left Cleft = (5.625-3.125) × 8 i.e. 20 MB. Similarly, in R3 the location P_3 is the starting location and the computation left Cleft =(5.2-5)×5 i.e. 1 MB of computation.

According to the proposed algorithm, T2 should be allocated to that resource where the computation left is minimum. Therefore, T2 will be allocated to R3 and the position of allocation is P_3 location i.e. in the revised schedule T2 is the first task allocated to R3 and all previously allocated tasks will be rescheduled one position next. For task T1, the same procedure is followed and it will be assigned to resource R2. So as per the new schedule, both the failed tasks will be completed before their respective deadlines.

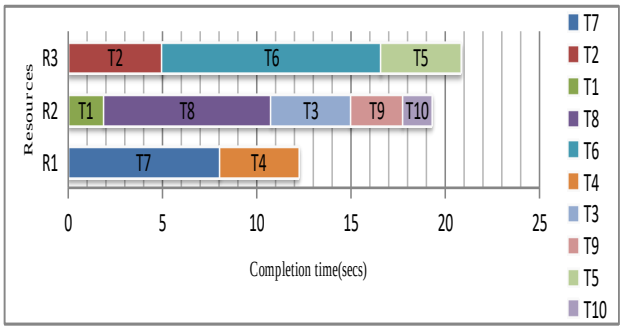

Fig 6. Generated schedule using Algorithm 2

Figure 6 presents the resultant schedule of the proposed algorithm2. In this schedule the tasks T1 and T2 are meeting their deadlines and thus the objective of meeting deadlines of all tasks is achieved and all resources within a cloud are utilised fully so that there is no need of migrating the tasks to another cloud. But, it is observed that the resources are not used evenly i.e., resources R2 and R3 are allotted with more tasks as compared to resource R1. It is also observed that makespan of R3 is 20.8 where as makespan of R1 is 12.1. So the resources are not utilised uniformly and Algorithm 3 is applied.

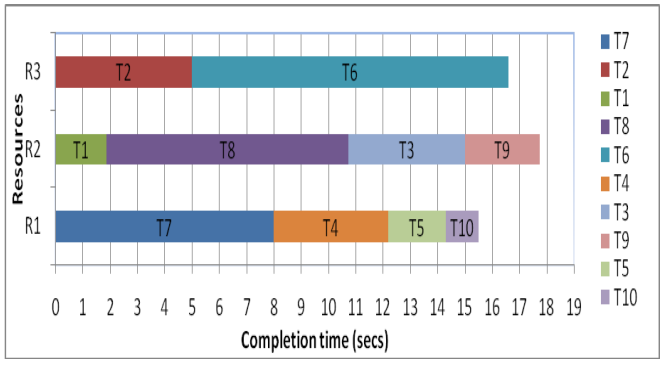

Fig 7. Generated schedule using Algorithm 3

The proposed algorithm 3 generated the resultant schedule as given in Figure 7. It can be observed from the Figure 7 that all the three resources are evenly used and the overall makespan is reduced from 20.8 to 17.75 seconds. Thus, it is observed that all the tasks are completed in lesser time.

V. CONCLUSION AND FUTURE SCOPE

Task scheduling is one of the major challenges in the framework of cloud environment in order to meet the deadlines as per the service level agreements. Another aspect that is equally important from the cloud provider's point of view is to balance the load across the available resources for effective resource utilization. In this paper, we have proposed two algorithms in order to deal with the deadline requirements as well as load balancing of resources. The algorithms are theoretically analyzed to show their efficacy. But, in this work we have not considered the dependencies that may exist among the tasks which may further complicate task scheduling. We would like to consider such aspects in our future scope of the work.

REFERENCES

1. Lee Badger, Tim Grance, Robert Patt-Corner and Jeff Voas. NIST Special Publication 800-146, *Cloud Computing Synopsis and Recommendations*, May 2012.
2. Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Mesina, Lee Badger and Dawn Leaf NIST Special Publication 500-292, NIST *Cloud Computing Reference Architecture*, Sep 2011
3. Barrie Sosinsky, Cloud Computing Bible WILEY INDIA Pvt. Ltd ISBN:9788126529803 Reprint:2012
4. Braun, Tracy D, et al. "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems" *Journal of Parallel and Distributed computing* , Volume 61, Issue 6, Pages 810 – 837, 2001
5. Freund R. F., Gherrity M., Ambrosius S., Halderman M., Hensgen D., Campbell M. , Keith E., Kidd T., Kussow M., Lima J. D., Moore L., Mirabile F., Siegel H. J. and B. Rust. "Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet". *7th IEEE Heterogeneous Computing Workshop* (HCW '98), pp. 184-199, 1998.
6. G. K. Kamalam and V. Murali Bhaskaran. "An Improved Min-Mean Heuristic Scheduling Algorithm for Mapping Independent Tasks on Heterogeneous Computing Environment", *International Journal Of Computational Cognition* (http://www.ijcc.us), Vol. 8, No. 4, December 2010.
7. Huankai Chen, F Wang, Na Helian and Gbola Akanmu " User-Priority Guided Min-Min Scheduling Algorithm For Load Balancing in Cloud Computing" *IEEE National Conference on Parallel Computing Technologies*,2013.
8. El-Sayed T. El-kenawy, Ali Ibraheem El-Desoky and Mohamed F. Al-rahamawy. "Extended Max-Min Scheduling Using Petri Net and Load Balancing" in *International Journal of Soft Computing and Engineering* (IJSCE) ISSN: 2231-2307, Volume-2, Issue-4, September 2012
.9. Yazir, Y.O.; Matthews, C.; Farahbod, R.; Neville, S.; Guitouni, A.; Ganti, S.; Coady, Y., "Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis," *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, July 2010 doi: 10.1109/CLOUD.2010.66
10. Etminani K. and Naghibzadeh M., "A Min-Min Max-Min selective algorithm for grid task scheduling,"*3rd IEEE/IFIP International Conferencein Central Asia on Internet, 2007*, Sept. doi:10.1109/CANET.2007.4401694.
11. S. Nagadevi, K. Satyapriya, Dr. D.Malathy, "A survey on economic cloud schedulers for optimized task scheduling", International Journal of Advanced Engineering Technology,2013
12. Tracy D Barun, Howard Jay Siegel, Noah Beck, "Acomparision of eleven static heuristics or mapping a class of independent task onto heterogeneous distribute computing systems", Journal of Parallel and Distributed Computing 61, pp. 810-837,2001
13. Chandrasekhar S Pawar, Rajnikant B Wagh, "Priority based dynamic resources allocation in cloud computing with modified waiting queue", International Conference on Intelligent Systems and Signal Processing, ISSN- 978-1-4799-0317-7,2013
14. Jiayin Li, Meikang Qiu, Jian-Wei Niu, Yu Chen, Zhong Ming, "Adaptive resource allocation for preemptable jobs in cloud systems", International Conference on Intelligent Systems Design and Application, ISSN- 978-1-4244-8136-1,2010

15. Huankai Chen, Frank Wang, Na Helian, Gbola Akanmu, " User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing", National conference on Parallel Computing Technology, ISBN- 978-1-4799-1589-7, 2013
16. Jiayin Li, Meikang Qiu, Zhong Ming, Gang Quan, Xiao Qin, Zonghua Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems ", Journal of Parallel and Distributed Computing, 72, pages 666-677, 2012.

AUTHORS PROFILE

Harish Kumar Patnaik is an Assistant Professor at KIIT University, Bhubaneswar, India. and he is currently pursuing his Ph.D. His current research focuses on scheduling on cloud environment and also works with web technology.

Manas Ranjan Patra holds a Ph.D. in Computer Science from the Central University of Hyderabad, India. Currently he is the Professor and Head in the Department of Computer Science, Berhampur University. He has been actively engaged in teaching and research in different areas of Computer Science.