Dynamic Task-Scheduling in Grid Computing using Prioritized Round Robin Algorithm

Sunita Bansal[1], Bhavik Kothari[1], Chittaranjan Hota[2]

[1]Computer Science & Information Systems Group
Birla Institute of Technology & Science
Pilani, Rajasthan, 333031, INDIA

[2]Computer Science & Information Systems Group
Birla Institute of Technology & Science, Pilani
Hyderabad Campus, Hyderabd, AP, INDIA

Abstract

Over the years, grid computing has emerged as one of the most viable and scalable alternatives to high performance supercomputing, tapping into computing power of the order of Gigaflops. However, the inherent dynamicity in grid computing has made it extremely difficult to come up with near-optimal solutions to efficiently schedule tasks in grids. The present paper proposes a novel grid-scheduling heuristic that adaptively and dynamically schedules tasks without requiring any prior information on the workload of incoming tasks. The approach models the grid system in the form of a state-transition diagram, employing a prioritized round-robin algorithm with task replication to optimally schedule tasks, using prediction information on processor utilization of individual nodes. Simulations, comparing the proposed approach with the round-robin heuristic, have shown the given heuristic to be more effective in scheduling tasks as compared to the latter.

Keywords: *Dynamic Scheduling, Grid Computing, Task Replica, Round Robin, Prioritized Round Robin, Prediction Information*

1. Introduction

Grid computing is a form of distributed computing, where a set of loosely coupled and heterogeneous computing nodes donate their unused processor cycles to create a pool of substantial processing capacity. In recent years, grid computing has emerged as one of the most feasible alternatives to process compute-intensive tasks, using co-operating processing nodes of ordinary capacity. The main advantage that grid computing offers, is that inexpensive computing nodes are coupled together to produce resource capacity comparable to high-end supercomputers, albeit at a lower cost. The major bottleneck that grids face is that individual processors might not be well connected to one another and thus, the model is more suited to applications which can be broken into several independent and atomic sub-tasks, without any requirement to communicate intermittent results between the grid nodes.

The principal challenge involved in any distributed computing environment, is that optimal scheduling of tasks dynamically entering the grid, becomes an NP-hard problem. Efficient grid scheduling is one of the key factors for achieving high performance in gird environments. Several heuristics [1, 2, 3, 4, 5] have been proposed in literature to schedule the tasks efficiently to the most suitable node present in the grid. A majority of these heuristics show satisfactory results in static grid environments. However, they cannot be directly applied in dynamic environments where tasks are continuously arriving in the grid at regular intervals.

The principal motivation of the present paper is to develop a scalable task-scheduling algorithm which can operate efficiently without the services of a full-fledged prediction system providing prior information on workload of incoming tasks. The paper proposes an enhancement of the existing round-robin heuristic [2], where we exploit information on the capacity of individual grid nodes, to prioritize tasks currently in execution, such that tasks currently allocated to slower machines are preferred for replication purpose over jobs executing on comparatively faster machines. The approach facilitates replication of tasks, hitherto assigned to execute on slower machines, on machines with higher processing capacity.

2. Related Work

There has been significant research in the past to study the classic problem of optimal job assignment in distributed environments such as grids. Heuristics, dedicated to scheduling tasks optimally in a grid environment, can be broadly categorized into the following classes.

A. Batch Mode Mapping Heuristics

Tasks are queued and collected into a set when they arrive in the batch mode [1]. They will be scheduled or mapped to their respective machines afterwards at a specific interval by the scheduling algorithm. In Min-min scheduling algorithm, each job will be always assigned to the resource which can complete it the earliest in order to spend less time completing all jobs. The Max-min scheduling algorithm is similar to Min-min scheduling algorithm except that it gives the highest priority to the job with the maximum earliest completion time. The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it.

B. On-line Mode Mapping Heuristics

Tasks are scheduled as soon as they arrive [1]. All the machines in the grid need to be referred to in order to decide the optimal machine that needs to be mapped to the incoming task. The MCT (minimum completion time) heuristic assigns each task to the machine that results in the task's earliest completion time. The MET (minimum execution time) heuristic assigns each task to the machine that performs the task's computation in the least amount of execution time. The mapping algorithm takes O(m) time (where m is the number of machines in the grid) to map a task to its optimal machine. If a task arrives in the Grid while the mapping algorithm is underway, it has to wait until the previous mapping is complete.

C. Ringed Round Robin Algorithm

The algorithm assumes no prediction information is available either for the task length or the processor capability [2]. Tasks arriving in the Grid are assigned to idle machines in a FCFS (First come Fist serve) manner. Task replication is performed later on, in a ringed round robin fashion, to achieve better resource utilization in the Grid. Both the task allocation and task replication routines are executed randomly without any basis or optimization criteria.

D. Scheduling Algorithm for Bag-Of-Tasks using Multiple Queues with duplication

The strategy of the algorithm is to schedule tasks according to their workloads and computing power of resources available [6]. The machines in the Grid are ranked according to their processing speed and are allocated tasks that are most compatible to their processing capacity. In addition, it adopts a duplication scheme in order to achieve optimal utilization of the machines available and to avoid undesirable scheduling decisions. The algorithm is static in nature as compared to the inherent dynamic nature of grid systems where tasks are continuously submitted to the system and get optimally scheduled on the machines available in the grid.

E. Self-Adaptive Scheduling System for Heterogeneous Computing

Ming Wu and Xian-He Sun [7] propose a prediction model based on probabilistic approach for long-term, application-level based task scheduling system in a distributed heterogeneous computing environment. The architecture of the distributed model comprises of a task allocator, scheduler and predictor, all integrated together, to form the GHS scheduling system. A self-adaptive task scheduling algorithm, based on probabilistic approach, is put forth to improve the accuracy of the GHS scheduling system for efficiently scheduling the meta-tasks.

F. Economic Task Replication for Scheduling in Distributed Systems

Amit Agarwal and Padam Kumar [8] attempt to minimize the number of task replications without affecting the overall makespan of the meta-task submitted to the grid, by proposing two workflow scheduling algorithms, namely, Reduced Duplication for homogeneous systems (RD) and Heterogeneous Economical Duplication (HED) for heterogeneous systems respectively. The proposed algorithms aim at optimizing the overall processor consumption, by removing some duplicated tasks in the schedule whose removal does not affect the makespan adversely, thereby producing scheduling holes in the system, which can, in turn, be used to schedule other distributed applications in the grid.

G. Task Duplication based Scalable Scheduling Algorithm for Symmetric Multiprocessors

Oh-Han Kang and Dharma P. Agrawal [11] present a task duplication based scalable scheduling algorithm for Symmetric Multiprocessors (SMP), referred to as the S3MP (Scalable Scheduling for SMP), to schedule the tasks of a DAG onto a bus-based SMP environment, facilitating duplication of certain critical tasks, so as to reduce the overall schedule length, by pre-allocating communication resources so as to avoid communication conflicts later on during scheduling.

H. Miscellaneous Heuristics

Bin Zeng et al. [3] propose a negotiation based model, where adaptive learning agents, representing individual resources and tasks, co-operate among themselves to help achieve a near-optimal schedule. N.Malarvizhi and V.Rhymend Uthariaraj [4] describe a scalable grid-architecture involving a Grid Resource Manager, assuming the role of a resource broker to select computational resources based on job requirements and the capacity of grid resources, so as to minimize the time to process each application along with transmission time associated with it. D. P. Spooner et al. [5] develop a multi-tiered scheduling architecture (TITAN) that uses a performance prediction system (PACE), along with brokers that are involved in distribution of jobs in the grid, to meet deadlines and significantly increase the

efficiency of resource utilization. The paper [9] presents a novel load balancing approach in a heterogeneous distributed environment. The scheduler takes into account the threshold value, based on the ratio of service rates, along with the queue length to determine whether it is beneficial to migrate a given local task to another node in the system or not. Markov process model is used to describe the behavior of the heterogeneous distributed system under the proposed policies. Ruay-Shiung Chang et al. [10] propose an Adaptive Scoring Job Scheduling algorithm (ASJS) for a distributed grid environment to reduce the completion time of submitted jobs, by assigning jobs to resources after looking into recent scheduling history of every available resource and then choosing the most optimal one. Computing intensive jobs and data intensive jobs are handled differently, and local and global updates are used to obtain the most recent status of grid resources to schedule jobs more effectively in real time. System ModelSyed Nasir Mehmood Shah et al. [12] propose an algorithm for CPU scheduling of a modern multiprogramming operating system, design and development of new CPU scheduling algorithms (the Hybrid Scheduling Algorithm and the Dual Queue Scheduling Algorithm) with a view to minimize overall task schedule. The following paper extends this prioritized round robin heuristic from a single system multiprogramming environment, onto a multi-processor distributed architecture.

3. System Model

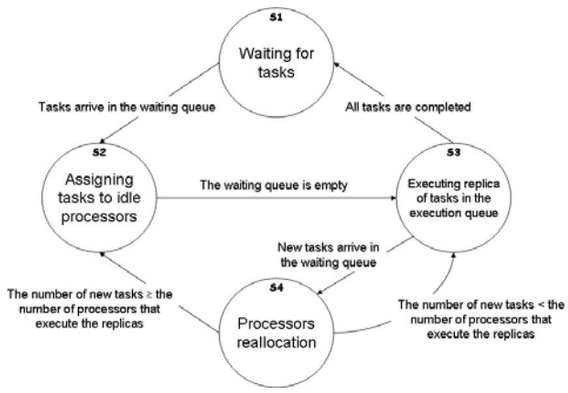
We consider a Grid system comprising of distributed computing nodes, and a central server for task allocation purpose. The Grid is modeled in the following state transition diagram, to be oscillating between the 4 given states. The system comprises of two queues to store records of the tasks currently in the Grid, namely the Waiting Queue and the Execution Queue. The Waiting Queue comprises of tasks in the Grid which are yet to be mapped to their respective machines, while the Execution Queue contains all the tasks which are currently in execution on at least one of the machines in the Grid.

The proposed model works under the following assumptions:

1. The model assumes that the tasks arriving in the Grid are atomic (cannot be broken into further sub-tasks) and are independent of one another.
2. It is assumed that the transportation costs involved, when tasks are mapped to their respective machines, are considered to be negligible.
3. It is assumed that we have no information available on the workload of the incoming tasks as it is not practically feasible to derive information regarding the same without the services of a full-fledged Prediction System.
4. The approach assumes that the processing speed of individual computing nodes is available to us. The initial processing speed of nodes is provided and the processing capacity of machines is then updated from time to time on the basis of the last application executed (workload) and the time taken.

4. Proposed Solution

We represent the heuristic as a state transition diagram (Given at fig-1.) with the grid occupying one of the four states at any given time. The waiting queue, comprising of tasks waiting to be mapped and executed on their respective machines, is implemented as a First in, First out (FIFO) queue where the task with the earliest arrival time is at the head of the queue and allocated an idle machine before other tasks waiting in the queue. The execution queue, consisting of tasks currently in execution, is implemented as a circular queue where each task in the queue has a specific order, and no task has the same order as any other task. The Execution Queue makes use of three pointers to scan the circular list in a Ringed Round Robin fashion. The *current pointer* is used to point to the task currently having the highest priority in the Execution Queue. The *next pointer* is used to point to the task with the second highest priority, which is nothing but the task lined next to the current task, one step in the clockwise direction. The *last pointer* is used to point to the task with the least priority in the Execution Queue, which is precisely the task placed besides the current task, a step in the anti-clockwise direction. The following is the description of the four states occupied by the grid system, during the course of time.



State I

Initial phase: State I is represented by an idle scheduler waiting for tasks to arrive in the grid. Incoming tasks are lined up in the waiting queue. Both the Waiting Queue and the Execution are initially empty. When the number of tasks in the Waiting Queue becomes more than the threshold value, a transition is made to State II.

State II

Initial phase: The Execution Queue is initially empty while the Waiting Queue comprises of a number of incoming tasks in the Grid. We maintain a list comprising of idle machines in the system. Initially all the machines are idle in State II, and hence the list contains all the machines in the Grid.

The tasks from the head of the Waiting queue and mapped one by one to the machines in the idle list. As soon as a task from the Waiting queue gets mapped to a machine, the given machine is subsequently removed from the idle list, while the task is

removed from the head of the Waiting Queue and inserted into the Execution Queue as explained in the System Model. The logic behind the mode of insertion is to give the highest priority to a task which has been assigned the slowest processor and vice-versa. The task with the highest priority in the Execution Queue would be the first one to get replicated because it is essentially the task with the slowest processors dedicated to it and hence replicating such a task would lead to a very high probability of the new machine executing the task before the machines already assigned to it.

State III

Initial phase: The waiting Queue is initially empty while the Execution Queue consists of tasks that are in execution in the Grid.

If a machine completes the execution of a task, the processor list of that particular task is referred to, and all the machines dedicated to executing the given task are released and made free while the task is removed the Execution Queue and the current, next and last pointers are updated if required. We update the processing power of the newly freed machine based upon the number of instructions that the machine executed for the previous task and the time it took to finish its execution. If the processing speed of the machine is greater than that of the *maxProcSpeed* of the task pointed to by the current pointer, then the given machine is required to execute the replica of the task pointed to by the current pointer.

Also, the current, next and last pointers are updated as follows. The current pointer becomes the last pointer, the next pointer becomes the current pointer and the next pointer would now be pointing to the task that was one step in the clockwise direction to the task pointed to by the erstwhile next pointer. Also, if the machine assigned to execute the replica of a task has a processor speed greater than that of *maxProcSpeed* of that task then the value needs to be updated to the processing speed of the given machine. We also keep track of the number of machines executing replicas and the number of tasks in the Waiting Queue, the information of which is exploited in State IV of the heuristic. At this point of time, we also check if there are any other idle machines present in the Grid from the idle list and assign tasks from the Execution Queue in the same way as described above, one by one, to these idle machines which are then subsequently removed from the idle list.

However, if a machine is found to have its updated processing speed to be less than that of the maxProcSpeed of the current task, then we do not assign the machine to execute the task replica for the simple reason that in all probability, the task would be accomplished faster on one of the machines already assigned to it as compared to the given machine. The machine is then inserted at the tail of the list containing the idle machines present in the Grid.

There are three scenarios, eventually possible, in State III.

Case I*:* All the tasks in the Execution Queue are successfully completed while the Waiting Queue is still empty. In this case, we traverse back to State I.

Case II*:* All the tasks in the Execution Queue are successfully completed while the number of tasks in the Waiting Queue is still less or equal to the threshold. In this case, we traverse back to State II.

Case III: The number of tasks in the Waiting Queue has exceeded the threshold before all the tasks in the Execution Queue could be completed. In this case, we traverse to State IV.

State IV

Initial Phase: Both the Waiting and the Execution Queue are initially non-empty. Two scenarios are possible at this point of time.

Case I: The number of machines executing replicas is less than the number of tasks in the Waiting Queue.

Case II: The number of machines executing task replicas is more than the number of tasks in the Waiting Queue.

The tasks in the Execution Queue are traversed in an anti-clockwise manner one by one, starting from the task pointed to by the last pointer (the least priority task) and if a task has more than one machine allocated to it, the machine at the tail end of the processor list is taken out of the list, freed from the task it was currently executing and assigned the task at the head of the Waiting Queue (after removing the task from the queue).

In Case I, we stop the traversal as soon as all the tasks in the execution queue are being run on one and only one machine and transition to State II. In Case II, we stop traversing the linked list as soon as all the machines in the Waiting Queue are assigned a machine, and then subsequently transition to State III.

5. Simulation

Simulation consists of a grid network comprising of a fixed number of computing nodes. Both, the number of machines as well as their processing capacity, are chosen randomly over a pre-defined range. Tasks enter the grid at random intervals within the range of 10 units and are assigned to the scheduler for allocation as soon as they arrive.

The simulation code defines one time unit as a single iteration, where we check for the current state of the grid system and accordingly perform the amount of work the scheduler can do in a single time unit when the grid is in that particular state. We run the simulation for as many steps as required to complete all the jobs submitted to the grid over the span of time. For our simulation purpose, we have considered the four test cases based on the degree of heterogeneity in node capacity and task workload entering the grid. In each case, we have considered 5 input cases with the following number of tasks-40, 80, 120, 160, 200. The following the four test cases in consideration. We have considered the overall turnover time as our yardstick to test the performance of the proposed heuristic against the round-robin algorithm.

Case I*:* Low heterogeneity in both processing capacity and

task workload,

Case II: High heterogeneity in processing capacity and low heterogeneity in task workload.

Case III: Low heterogeneity in node capacity and high heterogeneity in task workload.

Case IV: High heterogeneity in both node capacity and task workload.

In Case I, since the heterogeneity in node capacity is low , our proposed approach does not perform significantly better than the existing round-robin heuristic. The following graph compares the performance of round-robin heuristic and the proposed approach.

In Case II, though there is low heterogeneity in the task workload in the gird, but since the node capacities vary over a comparatively wide range, we observe significant improvement in results when we compare our approach with that that of the former heuristic.

In Case III, though there is high heterogeneity in the task workload but without a high heterogeneity in the node capacity in the gird, again, the algorithm does not show much improvement over the round-robin algorithm.

In Case IV, due to high variation in individual node capacities as well as in workloads of tasks arriving in the gird, we see a marked improvement of our heuristic in comparison the round-robin algorithm as high variation in processing speeds means that we can exploit faster available machines to replicate tasks which are currently running on comparatively slower machines.

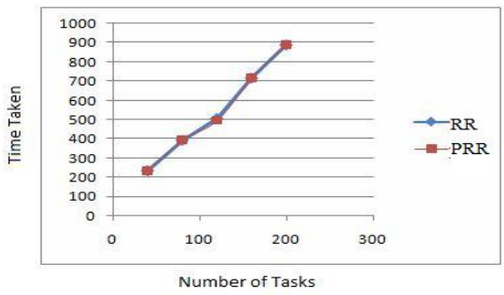


Fig. II. Compares RR(Round Robin) and PRR (Prioritized Round Robin) for Case I.

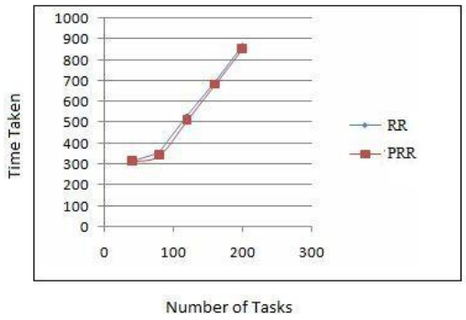


Fig. III. Compares RR with PRR for Case II.

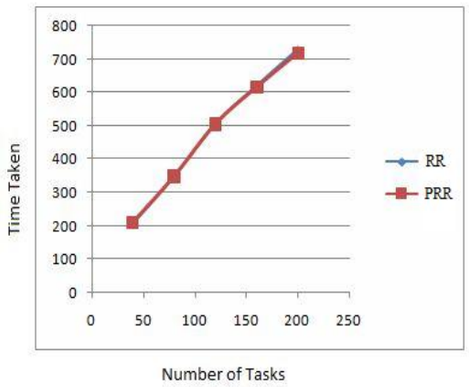


Fig. IV. Compares RR and PRR for Case III.

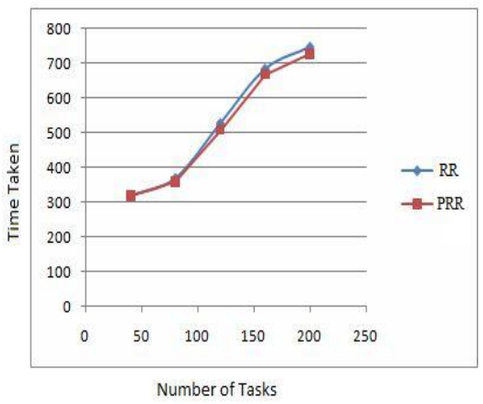


Fig. V. compares RR and PRR for Case IV.

6. Conclusions

The following paper describes a novel approach to schedule tasks efficiently in a grid environment, without having prior information on workload of incoming tasks. We propose an enhancement to the existing round-robin heuristic by prioritizing tasks eligible for replication. The approach is based on exploiting information on processing capability of individual

grid resources and applying replication on tasks assigned to the slowest processors. Future work will focus on designing a scalable architecture for scheduling tasks on grids comprising of nodes of the order of thousands, helping us to assess the performance of the heuristic to a better extent. In the present approach, we have ignored the costs involved in transporting tasks to their designated machines. To take the communication costs involved into account, along with the obvious processing costs of these tasks, is intended to be taken up as a subject for our future work.

References

[1] Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In: Proceedings of the HCW '99 Proceedings of the Eighth Heterogeneous Computing Workshop, pp. 30-30, IEEE Computer Society Washington, DC, USA (1999)

[2] Lee, L.T., Liang, C.H., Chang, H.Y.: An Adaptive Task Scheduling System for Grid Computing. In: Proceedings of the Sixth IEEE conference on Computer and Information Technology, CIT'06, pp. 57-57, IEEE Computer Society (2006)

[3] Zeng, B., Wei, J., Liu, H.: Dynamic Grid Resource Scheduling Model Using Learning Agent. In: Proceedings of the 2009 IEEE International Conference on Networking, Architecture, and Storage, NAS'09, pp. 67-73, IEEE Computer Society Washington, DC, USA (2009)

[4] Malarvizhi, N., Uthariaraj, V.R.: A Minimum Time To Release Job Scheduling Algorithm in Computational Grid Environment. In: Proceedings of the Fifth International Joint Conference on INC, IMC AND IDC, 2009, NCM'09, pp. 13-18, IEEE Computer Society (2009)

[5] Spooner, D.P., Jarvis, S.A., Caoy, J., Sainiz, S., Nudd, G.R.: Local Grid Scheduling Techniques using Performance Prediction. IEE Proceedings-Computers and Digital Techniques 150 (2) (2003) 87–96

[6] Lee, Y.C., Zomaya, A.Y.: Scheduling Algorithm for Bag-Of-Tasks using Multiple Queues with duplication. In: Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse (ICIS-COMSAR'06), pp. 5-10, IEEE Computer Society(2006).

[7] Wu, M., Sun, X. H.: A General Self-adaptive Task Scheduling System for Non-dedicated Heterogeneous Computing. In: Proceedings of the Fifth IEEE International Conference on Cluster Computing, 2003 (CLUSTER'03), pp. 354-354, IEEE Computer Society(2003)

[8] Agarwal, A., Kumar, P.: Economical Duplication Based Task Scheduling for Heterogeneous and Homogeneous Computing Systems. In: Proceedings of the Advance Computing Conference, 2009(IACC' 09), pp. 87-93, IEEE Computer Society(2009)

[9] Wang, J.L., Lee, L.T., Hunag, Y.J.: Load Balancing Policies in Heterogeneous Distributed Systems. In: Proceedings of the 26th Southeastern Symposium on System Theory, 1994, pp. 473-477, IEEE Computer Society(1994)

[10] Chang, R.S., Lin, C.Y., Lin, C.F.: Scheduling Jobs in Grids Adaptively. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, 2009, pp. 19-25, IEEE Computer Society

[11] Kang, O.H., Agrawal, D.P.: S3MP: A Task Duplication Based Scalable Scheduling Algorithm for Symmetric Multiprocessors. In: Proceedings of the 14th International Parallel and Distributed Processing Symposium, 2000 (IPDPS 2000), pp. 451-456, IEEE Computer Society(2000)

[12] Shah, S., Mahmood, A., Oxley, A.: Hybrid Scheduling and Dual Queue Scheduling. In: Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology, 2009 (ICCSIT 2009), pp. 539-543, IEEE Computer Society (2009).

Sunita Bansal received the M Sc (Computer Science) and M Tech (Computer Science) degrees from the Banasthali Vidyapith, Banasthali of India in 2003 and 2005, respectively. After completed her M Tech she joined Mody Institute of Technology and Science, Laxamangarh (Sikar) of India in July, 05 and left Sept, 05 and join Birla Institute of Technology & Science, Pilani of India. She is currently a Ph D scholar and faculty in Birla Institute of Technology and Science, Pilani of India. She is nuclear member and web administrator of Research and Consultancy Division, Birla Institute of Technology and Science, Pilani of India. Her papers are published in Springer-Verlag, Berlin, Heidelberg, IEEE and Academy Publishers, Finland. She is the Life member of the Computer Society of India, Indian Society for Technical Education, New Delhi (India), The Indian Science Congress Association, Kolkata (India) and International Association of Engineers (IAENG), USA and International Association of Computer Science and Information Technology, Singapore.

Bhavik Kothari received Bachelor degree in Computer Science from Birla Institute of Technology and Science, Pilani of India in 2010.

Chittaranjan Hota is currently Associate Professor and Head, Computer Science and Information Systems department at Birla Institute of Technology and Science, Pilani Hyderabad Campus, Hyderabad. He is with BITS, Pilani since the year 2000. He was the Faculty In-Charge of Information Processing and Business Intelligence Unit at BITS-Hyderabad for a period of first two years where he was instrumental in Campus Network design and implementation. He has worked in various Indian universities at different levels over past 20 years. He had several visiting researcher and visiting professor assignments (from two months to a semester at each place) at International academic and research institutes like, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia; Helsinki Institute of Information Technology, Helsinki, Finland; Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, Helsinki, Finland, and City University, London over past several years. He has published extensively in national and international conferences and journals. He is a life member of ISTE, and IE, India. His research interests are in the areas of Traffic Engineering in IP Networks, Security and Quality of Service issues over the Internet, Peer-to-Peer Overlays, Mobile Wireless Networks, and Cloud Computing