ijournals International Journal of Software & Hardware Research in Engineering
ISSN No: 2347-4890 Volume 2 Issue 1/ January 2014

EDZL SCHEDULING OF REAL-TIME TASKS IN DISTRIBUTED SYSTEMS

Manojkumar.R[1], Vinoth Kumar.S[2], Dr. Karthik.S [3]

PG Scholar, Department of CSE, SNS College of Technology, Coimbatore-35[1].
Assistant Professor, Department of CSE, SNS College of Technology, Coimbatore-35[2].
Head and Dean, Department of CSE,SNS College of Technology,Coimbatore-35[3].

Abstract- *Scheduling plays an important role in any multiprocessor system. Here we are going to introduce a optimal scheduling technique in multiprocessor system.Various subset of scheduling algorithm ranging from Classic Rate Monotonic Scheduling Algorithm to the Real Time Self Adjusting Dynamic Scheduling Algorithm for to Earliest Deadline first until Zero-Laxity (EDZL). Each algorithm is evaluated on the design choices made and their applicability to the problem at hand. It presents detailed look into the Earliest Deadline first until Zero-Laxity (EDZL) scheduling algorithm has received growing attention thanks to its effectiveness in multicore real- time scheduling. EDZL was used in real-time to satisfy all task deadlines, when the total utilization demand does not exceed the utilization capacity of the processors in distributed systems.*

Index Terms- EDZL,Real Time System,Scheduling.

I. Introduction

A Real-Time System (RTS) is defined as a system in which the time where the outputs are produced is significant. In RTS within a specified given time bound which is also term as deadline, the output must be produced. The logical result and the time in which the result was produced defined the correctness of RTS. The system may enter an incorrect state if a correct result is produced too early or too late with respect to the specified time bounds or deadlines. Real-Time computing (RTC) deal with both the hardware and software systems that are mainly subject to a "real-time constraint".eg. Operational deadlines from an event to system response. Real-time programs guarantee response within a given strict time constraints defined, which is often referred to as "deadlines". The responses in Real Time System are often understood to be in the order of milliseconds, and sometimes microseconds also. Conversely, In a system guarantee of a reponse within a timeframe cannot be given without real-time facilities. i.e. regardless of actual or expected response times.

Real-time software may refer to use of the following: synchronous programming languages, real-time operating systems, and real-time networks, each of which provide essential frameworks on which to build a real-time software application.

A real-time system may be one where its application can be considered (within context) to be mission critical. The anti-lock brakes on a car are a simple example of a real-time computing system — the real-time constraint in this system is the time in which the brakes must be released to prevent the wheel from locking. Real-time computations can be said to have failed if they are not completed before their deadline, where their deadline is relative to an event. A real-time deadline must be met, regardless of system load.

Type of real time system

A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed. Real-time systems, as well as their deadlines, are classified by the consequence of missing a deadline.

Hard : Missing a deadline is a total system failure.

Firm: Infrequent deadline misses are tolerable, but may degrade the system's quality of service. The usefulness of a result is zero after its deadline.

Soft: The usefulness of a result degrades after its deadline, thereby degrading the system's quality of service.

Thus, the goal of a hard real-time system is to ensure that all deadlines are met, but for soft real-time systems the goal becomes meeting a certain subset of deadlines in order to optimize some application specific criteria. The particular criteria optimized depends on the application, but some typical examples include maximizing the number of deadlines met, minimizing the lateness of tasks and maximizing the number of high priority tasks meeting their deadlines.

Soft real-time systems are typically used where there is some issue of concurrent access and the need to keep a number of connected systems up to date with changing situations; for example software that maintains and updates the flight plans for commercial airliners. The flight plans must be kept reasonably current but can operate to a latency of seconds. Live audio-video systems are also usually soft real-time; violation of constraints results in degraded quality, but the system can continue to operate.

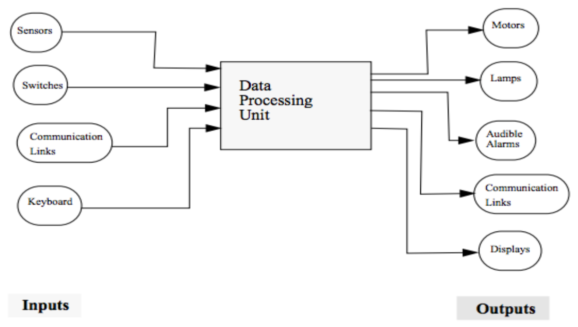

Fig: Real Time System

Real-Time Issues in Parallel and Distributed Computing

The easiest way to develop and run a parallel/distributed real-time system seems to take existing software, with proven operational record in a parallel or distributed environment, and run it for real time. This approach, however, raises the obvious question, how contemporary tools, suitable for traditional distributed applications, would perform in real time? In order to find an answer to this question, we invited researchers we knew would address exactly this problem.

Polze, Malek, Wallnau and Plakosh discuss in their invited contribution the use of off-the-shelf components based on CORBA, in the construction of distributed real-time software systems. They pinpoint several problems that arise with an intuitive simplistic approach, and develop their own solution to most of them. Their solution is based on what they call Composite Objects. The main idea of composite objects is to leave programming on the high level of abstraction with CORBA and to concentrate all issues related to the detailed knowledge of the objects' timing behavior with the special extension mechanisms. The introduction of Real-Time Computing with Off-the-Shelf Components: The Case for CORBA serves as a good background to the rest of this issue, because it outlines a number of problems that need fundamental solutions, not just quick fixes.

Real-Time Communication Protocols

The issues of scheduling and load balancing can never be fully resolved without thorough knowledge of communication protocols, which allow participating parties to exchange information. Real-time communication is particularly tricky due to uncertainties in the network, such as loss of messages or node failures, so that guaranteeing message delivery is difficult unless special measures are taken.

In this view, Tunali, Erciyes and Soysert discuss in their contribution, A Hierarchical Fault Tolerant Ring Protocol for Distributed Real-Time Systems, an approach to solve a communication problem in distributed real-time systems via the use of a synchronous communication protocol operating on hierarchical rings. The fault-tolerant algorithm developed allows the protocol to maintain communication in case of crash failures and is easily usable for real-time applications.

Gannod and Bhattacharya, in their paper titled Real-Time Multicast in Wireless Communication, present a new way to resolve multicast communication in wireless/mobile networks. In particular, they define a new performance metric for real-time multicast networks: delay of reconstruction of the multicast tree at the instance of node migration. This issue is important, because nodes in a mobile network tend to migrate from cell to cell causing the necessity to rebuild their descriptive patterns. In their paper, the authors give some simulation results of measuring performance metrics as a function of different system configurations when node migration occurs.

The third contribution addressing the development and application of communication protocols in the real-time domain, by Benzekri and Sarafoglou, is titled Protocols for Real-Time Network Applications Programming. The authors discuss the entire hierarchy of protocols (protocol stack) usable in real-time communication, in particular, IP Multicast, RTP/RTCP (Real-Time Protocol and Real-Time Control Protocol), and RSVP (Resource ReSerVation Protocol) and present related work on a video server application in a campus-wide network.

Real-Time System Development

One could think that having resolved the issues with real-time scheduling, load balancing, and real-time communication, would let the researchers and practitioners sleep quietly, but that is not so. Knowing what algorithms and protocols to use is just a tip of an iceberg—one needs to apply these concepts in real circumstances. In other words, intensive development and verification procedures are needed to prove that at least some of these concepts work.

In their contribution, Software Development and Verification of Dynamic Real-Time Distributed Systems Based on the Radio Broadcast Paradigm, van Katwijk, de Rooij, Stuurman and Toetenel address the use of an architectural style, defined as the Radio Broadcast Paradigm as a basis for software development. They develop formal verification techniques to analyze the required behavior of an underlying distributed system built according to this paradigm. Their principle relies on subsequent use of experimentation, abstraction and verification, and is labeled bottom-up as opposed to traditionally used top-down methodologies. They discuss the application of their verification approach to determining the timing constraints of an implemented communication protocol.

All theoretical methods are only as good as they get into practical use. Therefore we gave a very serious thought as to what particular parallel or distributed applications we would like to see covered in this issue. From practical research work we knew that there may exist only a few more challenging applications, in the development of parallel or distributed real-time systems, than those used in space research and highenergy physics. We were lucky enough to participate recently on Ph.D. defense committees of two practitioners who agreed to make invited contributions based on their dissertation work.

Scheduling algorithm in real time system.

The scheduling algorithm is of paramount importance in a real-time system to ensure desired and predictable behavior of the system.

A scheduling algorithm can be seen as a rule set that tells the scheduler how to manage the real-time system, that is, how to queue tasks and give processor-time. The choice of algorithm will in large part depend on whether the system base is uniprocessor, multiprocessor or distributed.

Uniprocessor system

A uniprocessor system can only execute one process at a time and must switch between processes, for which reason context switching will add some time to the overall execution time when preemption is used.

Multiprocessor system

A multiprocessor system will range from multi-core, essentially several uniprocessors in one processor, to several sep- arate uniprocessors controlling the same system.

Distributed system

A distributed system will range from a geographically dispersed system to several processors on the same board. In a distributed system the nodes are autonomous while in a multiprocessor system they collaborate somewhat more, but this line is not as clear cut as it may sound as similar communication delays will occur.

In real-time systems all tasks will have a deadline, an execution time and a release time. In addition there are other temporal attributes that may be assigned to a task. The three mentioned are the basic ones. The release time, or ready time is when the task is made ready for execution. The deadline is when a given task must be done executing and the execution time is how long time it takes to run the given task. In addition most tasks are recurring and have a period in which it executes. Such a task is referred to as periodic. The period is the time from when a task may start until when the next instance of the same task may start and the length of the period of a task is static.

An example, shown in Figure 2, of scheduling can be made using three tasks T1, T2 and T3 with execution time and deadline of (1, 3), (4, 9) and (2, 9) respectively and periods equal to their deadlines. These tasks can be scheduled so that all tasks get to execute before the deadlines

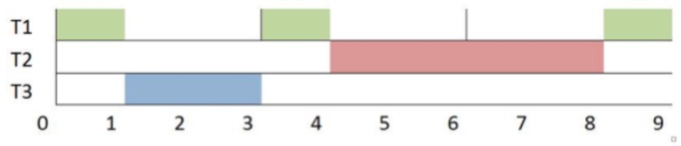

Fig 2: Scheduling task of T1,T2,T3

A system can consist of many tasks and the example above uses three periodic tasks, but there can also be aperiodic tasks which are tasks without a set release-time. These tasks are activated by some event that can occur at more or less any time or maybe even not at all. The scheduling example shows a minor system and the schedule can be made either before the system is activated, which is referred to as off-line scheduling, or during the running of the system and is then referred to as online scheduling. The example is very simple as it does not show priorities or use preemption. There are also other properties of interest when looking at scheduling.Properties a task may use briefly explained:

Release/ready time: The time a task is ready to run and just waits for the scheduler to activate it.

Deadline: The time when a task must be finished executing.

Execution/run time: The active computation time a tasks needs to complete.

Worst Case Execution Time (WCET): The longest possible execution time for a task on a particular type of system.

Response time: The time it takes a task to finish execution. Measured from release time to execution completes, including preemptions.

Priority/weight: The importance given a task in con- text of the schedule at hand.

Preemption is when a task that is executing on the processor becomes interrupted, its state is saved and then it is exchanged for another task. This switching of tasks on a processor is referred to as a context switch and takes a small amount of time each time it occurs. A preempted task will get to finish executing after the preempting task is done. Not all processors support preemption and algorithms can be divided into preemptive and non-preemptive scheduling algorithms. In real-time scheduling preemption is governed by priority.

A real-time system also have requirements based on dead- line, a real-time system can either be hard or soft depending on the consequences of missing a deadline. A hard real-time system is never allowed to miss a deadline because that can lead to complete failure of the system. A hard real-time system can be safety-critical and this means that if a deadline is missed it can lead to catastrophically consequences, which can harm persons or the environment. It is a crucial requirement that a task starts on time and do not miss the deadline; being not too early nor too late. In a soft real-time system a deadline is allowed to be missed, while there is no complete failure of the system it can lead to decreased performance.

Scheduling in multi-core processor unit

In multiprocessor scheduling, algorithms developed for uniprocessor scheduling can be applied on considering each core of the multiprocessor as an isolated core, which is a uniprocessor. However, in multiprocessor scheduling the difficulty of verifying that the execution of different tasks on multiple cores does not interfere with each other and also determining which tasks should be given to a certain core increases the complexity greatly compared to uniprocessor scheduling. Moreover scheduling algorithms for multiprocessors of- ten involves heuristics to simplify the task of finding a feasible schedule. The two main approaches for scheduling algorithms on multiprocessors are global scheduling and partitioning scheduling.

Global scheduling algorithms put ready tasks in a queue sorted based on priority. The task with currently highest priority, which is first in the queue, is selected by the scheduler and will execute on one of the processors and may be preempted or migrated if necessary.

In partitioning scheduling algorithms each task is assigned to one processor and will exclusively execute on that processor. This will result in that instead of being a multiprocessor problem it will be a set of uniprocessor problems. Uniprocessor algorithms can then be used for each processor, which is an advantage, but using non- optimal heuristic hard and usually solves the problem of partitioning the task to the processors. Also as shown by there exist systems, which are schedulable if and only if tasks are not, partitioned.

The myopic algorithm is a multiprocessor algorithm that schedules tasks not only for CPU time but also for shared resource requirements and uses heuristics to simplify the search for a feasible schedule at any given time. As a new task arrives it will be given to one of the processors, which will see if schedulability of that task can be guaranteed. If no schedule can be found the task will be sent on to another processor, while it will be kept if a schedule is found that can guarantee that task along with all currently existing tasks on that processor. When a new task is made ready this algorithm looks at:

• Arrival time TA

• Deadline TD

• Worst case processing time TP • Resource requirements TR

Premises for tasks are that they are independent, no periodic, non-preemptive and use resources in shared or exclusive mode. When scheduling a task the algorithm calculates the earliest start time, T_{EST} for the task based upon when the required resources will be available. The following condition is to be true for every task when a new schedule is made:

$$0<T_A <T_{EST} <(T_D\ T_P)$$

All tasks that are at any one point to be scheduled is placed in a list sorted according to deadline and a schedule is searched for by using a tree structure to find feasible schedules in which the root is an empty schedule and each leaf is a schedule while not every leaf is feasible. The tasks are inserted into the schedule, which then is one node level down in the tree, one at a time until a schedule is found or a deadline miss will occur. In the case of a miss there will be some back tracking in the tree and a new schedule will be looked at and so forth until a plausible schedule is found. Every time an- other task is to be inserted there is a heuristic function H, which looks at more than deadline, is called to evaluate a portion of the tasks in the list to find the most appropriate task to insert into the schedule at that point.

The original version of this algorithm looked at all tasks that were not in the schedule every time a task was to be added, but this algorithm only looks to the first few in a sorted list and this near sightedness is what makes the algorithm myopic. Compared to the non-myopic version of this algorithm and to other multiprocessor algorithms of its time the myopic algorithm was an efficient improvement. The myopic algorithm is O(nk) where 1< k < n depending on the number of tasks to be scheduled at the time as n equals all tasks to be scheduled and k is a subset there of.This then makes the myopic algorithm have O(n) while the original non-myopic algorithm was O(n^2).

II Existing Scheduling mechanism under RTS

In 1973 Liu and Layland published the paper [3], which provides the theoretical foundation to all modern scheduling algorithms for real-time systems. In their analysis, the authors limit themselves to independent, periodic, preemptable, hard real-time tasks executing on a single processor. The processor is always executing the task with the highest priority. Although Liu and Layland only address scheduling of real-time tasks on a single processor, their results turned out to be applicable to distributed systems as well. The RMS algorithm provides a good mathematical basis for determining the upper bound for processor utilization.

Real-time applications can be more efficiently scheduled on systems with extra or idle processors through task duplication. The Search and Duplication Based Scheduling algorithm (SDBS) [4] is a static scheduling algorithm designed to minimize the completion times of real-time applications by taking advantage of under-utilized processors. This algorithm produces efficient schedules for applications that are easily partitioned into tasks. Tasks are assumed to be non-preemptive processes running on a homogeneous, connected, and unbounded set of processors.

The SDBS algorithm performs three steps to determine an optimal schedule for the tasks of an application. Step one is to compute the earliest start time (EST) and earliest completion time (ECT) for each node in the task dependency DAG. The main focus of the algorithm is to identify critical tasks within the application.

The RT-SADS algorithm is designed for scheduling aperiodic, non-preemptable, independent, soft real- time tasks with deadlines on a set of identical processors with distributed memory architecture. The primary goal of the designers of this scheduling algorithm is scalability of deadline compliance with respect to increases in the number of processors and task arrival rate.

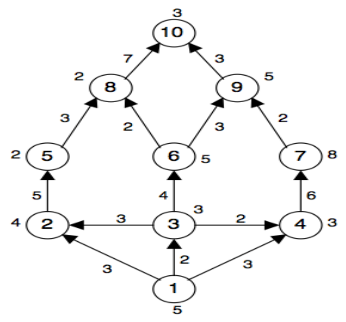

Fig 3:Search and Duplication based scheduling

RT-SADS uses a dedicated processor to perform incremental searches through an assignment-oriented task-processor space. Each scheduling stage j receives as input a set of tasks Batch(j) and produces as output Batch(j+1) and a feasible (although possibly only partial) schedule Sj. Batch(j+1) is formed from Batch(j) by removing all scheduled tasks and tasks with missed deadlines and adding tasks which arrived during stage j. At the end of any scheduling stage there usually more than one feasible partial schedule available. RT- SADS allows the use of a cost (utility) function for determining which of the available schedules to choose.

Another significant work is by Lee et al. (2003). They proposed an algorithm called "Maximum Workload derivative First with Fragment Elimination" (MWF- FE) for the on-line scheduling problem. This algorithm utilizes the property of scalable tasks for on-line and real-time scheduling. They assumed tasks are scalable if their computation time can be decreased (up to some limit) as more processors are assigned to their execution. They also defined the total amount of processors time devoted to the execution of a scalable task as the workload of the task. They assumed, as the number of processors allocated to a scalable task increases, its computation time decreases but its workload increases because of parallel execution overhead, such as contention, communication, and unbalanced load distribution. Managing processors allocated to the tasks as few as possible without missing their deadlines reduce In Lee et al. algorithm, the total workload of all scheduled tasks.

David B. Stewart and Pradeep K. Khosla proposed this algorithm [6]. The minimum-laxity- first algorithm assigns a laxity to each task in a system, and then selects the task with the minimum laxity to execute next. This algorithm is also called as Least Laxity First (LLF).Laxity is defined as follows:

Laxity = deadline_time - current_time - CPU_time_still_needed

Laxity is a measure of the flexibility available for scheduling a task. A laxity of tl means that even if the task is delayed by tl time units, it will still meet its deadline. A laxity of zero means that the task must begin to execute now or it will risk failing to meet its deadline. The laxity of a process is defined as the deadline minus remaining computation time. The algorithm gives the highest priority to the active job with the smallest laxity [7]. Then the job with the highest priority is executed. While a process is executing, another whose laxity has decreased to below that of the running process can preempt it. A problem arises with this scheme when two processes have similar laxities. One process will run for a short while and then get preempted by the other and vice versa. Thus, many context switches occur in the lifetime of the processes. The least laxity first algorithm is an optimal scheduling algorithm for systems with periodic real-time tasks. If each time a new ready task arrives, it is inserted into a queue of ready tasks, sorted by their laxities.

EDF assigns highest priority to jobs with earliest deadlines. EDF can be efficiently implemented and the total number of preemptions is bounded by the number of jobs [8]. However, the utilization bound of EDF can be very low on multiprocessor. Dhall et al. [9] showed that EDF has difficulty in scheduling task sets in which high utilization tasks and low utilization tasks are mixed.

Yi-Hsiung Chao proposed this algorithm [12]. EDF cannot guarantee all periodic tasks to meet their deadlines if the total utilization is greater than (m+1)/2 when the system has m processors. On the other hand, LLF has a good schedulability condition but has a high context-switching overhead and causes a large number of preemptions.

EDZL is a hybrid preemptive priority-scheduling scheme in which jobs with zero laxity are given highest priority and other jobs are ranked by deadline. In this report apply demand analysis to Earliest Deadline Zero Laxity (EDZL) scheduling, and derive a test that is sufficient to guarantee that a system of independent sporadic tasks with arbitrary deadlines will not miss any deadlines if scheduled by global EDZL on a platform with m identical processors. This survey also show through experiments how the new EDZL schedulability test compares to known schedulability tests for global Earliest-Deadline-First (EDF) scheduling, on large numbers of pseudo-randomly generated task sets.

EDZL has been studied by Cho et al. [8], who showed that when EDZL is applied as a global scheduling algorithm for a platform with m identical processors its ability to meet deadlines is never worse than pure global EDF scheduling, and that it is "suboptimal" for the two processor case, meaning that "every feasible set of ready tasks is schedulable" by the algorithm. They propose this weak definition of optimality as being appropriate for on-line scheduling algorithms, which cannot take into account future task arrivals. Cho et al. also provide experimental data, showing that even though EDZL is not "suboptimal" for m > 2, it still performs very well, and out-performs global EDF in particular, while not incurring significantly more context switches compared to global EDF.

Earliest Deadline first until Zero Laxity (EDZL) algorithm, which combines the EDF and MLF algorithms. EDZL schedules jobs based on their deadlines and laxities. When all jobs have positive laxities, EDZL schedules jobs according to EDF. Whenever the laxity of a job becomes zero, EDZL schedules the job with the highest priority. In EDZL scheduling, a scheduler must be invoked every time any job reaches zero laxity. Since the zero laxity can occur at any time, the number of scheduler invocations is not bounded. In addition, fine-grained timers are required to make the scheduling points for the priority promotions at zero laxity. Meanwhile the scheduling points of EDF and EDF-US[x] are made only at job releases and completions.

Earliest Dead- line Critical Laxity (EDCL), for the efficient scheduling of sporadic real-time tasks on multiprocessors systems. EDCL is a derivative of the Earliest Deadline Zero Laxity (EDZL) algorithm in that the priority of a job reaching certain laxity is imperiously promoted to the top, but it differs in that the occurrence of priority promotion is confined to at the release time or the completion time of a job.

This modification enables EDCL to bound the number of scheduler invocations and to relax the implementation complexity of scheduler, while the schedulability is still competitive with EDZL. The schedulability test of EDCL is designed through theoretical analysis. In addition, an error in the traditional schedulability test of EDZL is corrected. Simulation studies demonstrate the effectiveness of EDCL in terms of guaranteed schedulability and exhaustive schedulability by com- paring with traditional efficient scheduling algorithms. Ear- liest Deadline Critical Laxity (EDCL), for the efficient scheduling of sporadic real-time tasks on multiprocessor systems. The basic concept of EDCL is that the priority of a job does not necessarily have to be promoted at zero laxity to meet its deadline but can be promoted at the re- lease time or the completion time of a job. This modification of the priority promotion concept leads to that a scheduler needs to be invoked only at job releases and completions. Thus, the number of scheduler invocations is bounded and the implementation complexity of scheduler is relax, while the schedulability is still competitive with EDZL.

III. CONCLUSION

In the simple case, scheduling may seem straightforward and is easy enough to understand, but once more tasks are added it becomes more troublesome to complete a schedule. Introducing several resources makes for more complicated schedules and increases complexity in the algorithms used to create these schedules. The requirements that real-time systems must fulfill are several and equally many are the approaches how to schedule such systems. As mentioned in this thesis optimal algorithms exist but their optimality is often only theoretical and not practical in actual systems.

Multiprocessor systems are the future as we see it now, but finding algorithms that takes full advantage of these systems is an arduous task in which much effort has been and is being made by researchers.

REFERENCES

[1] C. Liu and J. Layland, “Scheduling Algorithms for Multi-Programming in a Hard-Real-Time Environment,” J. ACM, vol. 20, no. 1, pp. 46-61, 1973.
[2] C.-G. Lee, K. Lee, J. Hahn, Y.-M. Seo, S.L. Min, R. Ha, S. Hong, C.Y. Park, M. Lee, and C.S. Kim, “Bounding Cache-Related Preemption Delay for Real-Time Systems,” IEEE Trans. Software Eng., vol. 27, no. 9, pp. 805-826, Sept. 2001.
3. L. Sha, T. Abdelzaher, K.-E. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A.K. Mok, “Real Time Scheduling Theory: A Historical Perspective,” Real-Time Systems, vol. 28, no. 2/3, pp. 101-155, 2004.
4. S.K. Lee, “On-Line Multiprocessor Scheduling Algorithms for Real-Time Tasks,” Proc. IEEE Region 10’s Ninth Ann. Int’l Conf., pp. 607-611, 1994.
5. J. Lee, A. Easwaran, I. Shin, and I. Lee, “Zero-Laxity Based Real- Time Multiprocessor Scheduling,” J. Systems and Software, vol. 84, no. 12, pp. 2324-2333, 2011.
6. S. Cho, S.-K. Lee, S. Ahn, and K.-J. Lin, “Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems,” IEICE Trans. Comm., vol. E85-B, no. 12, pp. 2859-2867, 2002.
7. M. Park, S. Han, H. Kim, S. Cho, and Y. Cho, “Comparison of Deadline-Based Scheduling Algorithms for Periodic Real-Time Tasks on Multiprocessor,” IEICE Trans. Information and Systems, vol. E88-D, pp. 658-661, 2005.
8. M. Cirinei and T.P. Baker, “EDZL Scheduling Analysis,” Proc. Euromicro Conf. Real-Time Systems, pp. 9-18, 2007.
9. T.P. Baker, M. Cirinei, and M. Bertogna, “EDZL Scheduling Analysis,” Real-Time Systems, vol. 40, pp. 264-289, 2008.
10. J. Goossens, S. Funk, and S. Baruah, “Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors,” Real-Time Systems, vol. 25, no. 2/3, pp. 187-205, 2003.
11.Y.-H. Chao, S.-S. Lin, and K.-J. Lin, “Schedulability Issues for EDZL Scheduling on Real-Time Multiprocessor Systems,” Information Processing Letters, vol. 107, pp. 158-164, 2008
12. S. Kato and N. Yamasaki, “Global EDF-Based Scheduling with Efficient Priority Promotion,” Proc. IEEE Int’l Conf. Embedded and Real-Time Computing Systems and Applications, pp. 197-206, 2008.

Manoj Kumar. R M.Sc. degree in Software Engineering from Annamalai University, Chidambaram, India in 2012. Currently, pursuing M.E degree in SoftwareEngineering from SNS College of Technology, Anna University, Chennai. His area of interest are software development, network security.

S. Vinoth Kumar received B.E degree in Computer Science and Engineering from Anna University, Chennai, India in 2006, M.E degree in Computer Science and Engineering from Anna University, Tirunelvelli, India in 2009. He is at present working as Assistant Professor in the department of Computer Science Engineering at SNS College of Technology, Coimbatore, India. His area of interest includes Image Processing, Network Security and Mobile Computing.

Professor Dr.S.Karthik is presently Professor & Dean in the Department of Computer Science & Engineering, SNS College of Technology, affiliated to Anna University- Coimbatore, Tamilnadu, India. He received the M.E degree from the Anna University Chennai and Ph.D degree from Ann University of Technology, Coimbatore. His research interests include network security, web services and wireless systems. In particular, he is currently working in a research group developing new Internet security architectures and active defense systems against DDoS attacks. Dr.S.Karthik published more than 35 papers in refereed international journals and 25 papers in conferences and has been involved many international conferences as Technical Chair and tutorial presenter. He is an active member of IEEE, ISTE, IAENG, IACSIT and Indian Computer Society.