International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-5, March 2019

Schedulability Analysis for Rate Monotonic Algorithm in Distributed Real-Time System

Leena Das, Durga Prasad Mohapatra

***Abstract:** Real-Time Monotonic algorithm (RMA) is a widely used static priority scheduling algorithm. For application of RMA at various systems, it is essential to determine the system's feasibility first. The various existing algorithms perform the analysis by reducing the scheduling points in a given task set. In this paper we develop a algorithm to compute the RMA schedulability in a distributed real-time system. In today's world all the high performance computation are done in some form of distributed systems. We look at the limitations of parallel system and show how a distributed system can overcome it and also propose a distributed algorithm for the Schedulability analysis.*

***Keywords:** distributed systems, real-time systems, RMA, Schedulability.*

I. INTRODUCTION

Real-time systems are different from other types of systems in the sense that they must provide accurate results in both temporal and logical aspects. Apart from being able to carry out the required task, the system must do it within a certain period of time. For example, when a temperature sensor in a thermal plant gives a warning, then the system must turn on cooling mechanisms within a certain interval of time barring which the plant's operation may fail. Here the response of the system must be correct as well as be completed in a given interval of time. A real-time system can be divided broadly into three spheres; the environment, the controller and the controlled object. The controller gets input from the environment and then provides information to the controlled object. The time it takes to provide the instruction is known as the execution time. The time after which an event repeats itself in the environment is known as the period of that event. Every event needs a certain time interval before which it has to be executed. This is known as the deadline. Thus a real-time system's primary goal is to provide a scheduling algorithm where all the deadlines are met, taking into account the period and execution time. To accomplish this there are a number of algorithms primarily categorized into two categories: static priority and dynamic priority algorithms. Static priority algorithms are those which the priority assigned are static in nature. While in dynamic, the priorities changes dynamically.

Manuscript published on 30 March 2019.
***Correspondence Author(s)**
Leena Das, Department of Computer Science, KIIT University, Bhubaneswar
Durga Prasad Mohapatra, Department of Computer Science, and Engineering, National Institute of Technology, Rourkela

This work concentrates on static priority scheduling, most specifically RMA. A distributed system uses software to coordinate tasks that are performed on multiple computers simultaneously. The computers interact to achieve a common goal, and they interact by sending each other messages. In any distributed algorithm, the actual calculations need to be broken up into separate elements that can be run on different computers. Some calculations may be entirely sequential. However, some calculations may be able to run in parallel and then combine the results. Distributed computing is used to solve complex computational problems that cannot be completed within a reasonable amount of time on a single computer. The time necessary to complete all the calculations is reduced by harnessing the power of multiple computers. In this paper we use the power of distributed computing to improve the performance of existing Schedulability analysis algorithm for RMA.

A. *Motivation*

Before the proper scheduling is carried out in a given sys-tem, it is of high importance that the task set is analyzed first. In most of the situations, the computation of Schedulability analysis requires less time than carrying out the actual Schedulability but gives valuable information regarding the nature of dataset. In this work, we implement a parallel algorithm for RMA Schedulability. In a multiprocessor environment there are number of processors which can be assigned work. Instead of running serial analysis in a single processor, by taking advantage of the processors, distributed computation can be done. The Improved Time Demand Analysis algorithm gives good enough performance but that can be improved in case of multiprocessor environment and we explore that.

B. *Objective*

The objectives of this paper are

1) To implement Improved Time Demand Analysis al-gorithm in a distributed multiprocessor system.
2) To show that distributed execution in a multiprocessor environment performs better than parallel execution.

C. Section Organization

This paper is divided into 5 sections. The first sections gives the motivation and objectives of the work. Section 2 discusses the basic concepts of real-time and distributed systems that are needed to understand the work. Section 3 presents the various literature survey done.

Section 4 presents the parallel algorithm and discusses the results obtained. In section 5 we present the conclusion and future work.

II. BASIC CONCEPTS

Rate Monotonic Algorithm (RMA) is one of the most widely used and effective scheduling algorithms. RMA uses mathematical model of static priority based scheduling where the priority is the period of the tasks. It supports the intuition that the tasks that occur more frequently should be given higher priority. RMA was first proposed in [1]. Working of RMA depends on the periods of the tasks. For a given task set to be termed as feasible, it must be possible to be scheduled with a given algorithm, in this case, RMA. These feasibility tests are generally known as schedulability bounds. For an algorithm to work, it must be within certain limits. Schedulability bound provides this limit. The work in [1] gave an initial feasibility bound. It was improved by Seto and Lehoczy in [2], [3] by the use of a time demand analysis function to make it an exact feasibility test. Our work focuses on this aspect of RMA scheduling. Various types of other bounds were also proposed. The concept of the harmonic period was explored by Kuo et. al in [4] that showed the schedulability of tasks satisfying the harmonic conditions. This report now analyses the initial method proposed in [2], [3], often termed as time demand analysis and implements it. We then propose an algorithm to improve this implementation.

Real-time systems can be divided into three broad cate-gories

1) Hard real-time Systems
2) Firm real-time Systems
3) Soft real-time Systems

If the completion of a task is not achieved within the deadline in a Hard real-time system then a system failure occurs. While if the system still can function with some degradation in performance if the deadlines are missed then the system is termed as Soft real-time System. Firm real-time systems are in between Hard and soft real-time systems. In this paper we have considered hard real-time systems i.e. all the deadlines must be met. The event that determines a course of action is known as a task. On the occurrence of a task, the system does processing and responds accordingly. There are three types of task

1) Periodic Tasks
2) Aperiodic Tasks
3) Sporadic tasks

Periodic Tasks are the tasks that occur after a specified interval of time. For example, a sensor sending temperature data every 10 seconds. We say that this task is periodic with a period of 10 secs. On the other hand, an Aperiodic task is one where the task can occur after any amount of time after the occurrence of the last instance, except immediately. Sporadic tasks are aperiodic tasks where the repetition period can be zero. In our paper, we deal with periodic tasks only. Now we describe the various notations used in our paper.

C_i : Execution time of i_{th} task
T_i : Period of i_{th} task
U_i : Utilization of i_{th} task
w_i : Time demand function value
t : Current time point

Any other notations have been described as and when used.

RMA was first proposed by [1] in 1973 as an optimal scheduling algorithm for static priority task set. The priority was assigned to the periods of the tasks. For a task set T (T_1; T_2; :::; T_n) period(T_i) < period(T_j) =) priority(T_i) > priority(T_j)

For validating the feasibility of a task set by determining whether it is schedulable or not, a variety of tests has been developed. The first universal feasibility bound for all types of scheduling systems is given by

$$\sum_{i=1}^{n} U_i \leq 1 \quad (1)$$

The sum of utilization of all the tasks in the task set should be less than equal to 1. This gives the necessary upper bound for any scheduling algorithm including RMA. But this is not sufficient. We refer to this bound as schedulability bound 1. A tighter feasibility test was proposed in [1] which stated that a periodic static priority system is feasible if

$$\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \quad (2)$$

Where n is the total number of tasks in the task set. The value tends to ln(2) as n tends to 1. This shows that in any sufficiently large task-set, if the total utilization is less than 0.693, then it can be scheduled. This, however, is a sufficient condition only. Even if the total utilization remains greater than this bound, it can still be static feasible. To take into account this factor, various necessary and sufficient tests were proposed [2], [6], [16], [17], [18], [26], [27]. The initial test proposed in [2], [6] was further improved in [16], [18]. These all tests remained pseudo-polynomial. The proposed tests can be divided into two types; iterative [6], [26] and as-per-task basis [2], [18], [19], [20], [21], [22]. The later analyses feasibility on task arrival times known as scheduling points. The former uses an iterative technique. Recently the work by Allah et. al. in [17] improved the work by Bini and Buttazzo in [16] by restricting the actual number of scheduling points. We now discuss the exact feasibility test upon which we propose our improvement.

A. Exact Schedulability Test

Phase I of a task is defined as the time when the first instance of the task is released into the system. Two tasks T_i and T_j are said to be of same phasing iff $I_i = I_j$.
In the work [1], [24], [25] it was shown that in the scenario where the phasing of all the tasks is equal, it results in the longest response time.

This is generally known as the critical instant. This scenario creates the worst case task set phasing and thus can be used as a criterion for the schedulability of the given task set. The idea can be re-framed as "A periodic task set can be scheduled by a fixed priority scheduling algorithm if the deadline of the first job of each task is met while using that algorithm from a critical instant". Since RMA is a fixed priority scheduling algorithm, the condition satisfies for it. Let T be a task set T (T_1; T_2; :::; T_n) with tasks having increasing periods (thus decreasing priority). As per RMA's priority property, a task having lower period has to be sched-uled before the task of a higher period. Thus a task T_i can only be affected by the tasks T_j where period(T_j) > period(T_i). This gives the intuition that while checking for the schedule ability of the task only that task should be considered whose periods more than the current (or priority less). This ordering can be achieved by sorting the task set in ascending order of their period. When the task set is now processed, the tasks are encountered with decreasing priority. As the At the time of critical instance, a value is calculated. This value gives the estimate as to how much the system is feasible. These ideas were stated mathematically in [2] as follows

$$wi(t) = Ci + \sum_{1}^{i-1} \lceil t/Tk \rceil * Ck \tag{3}$$

$w_i(t)$ is the time demand function of ith task when it is released at the critical instant. As can be seen, the tasks from starting to i 1 only contribute to this value function. The tasks after the ith task cannot affect as they have lower priority (as task set is sorted). After this calculation, the schedule ability bound was given as

$$w_i(t) < t \tag{4}$$

Where $w_i(t)$ can be interpreted as demand and the time t can be seen as the supply. So, the demand must be less than the supply for the task set to be feasible. Equation (4) involves checking for time instances the demand of a given task. The time instances that must be checked relates to the tasks having higher priority than the current. As mentioned earlier, the property of RMA asserts that only higher priority tasks can affect the current task. Thus only those time instances need to be checked that are multiples of the period of all the high priority tasks.

$$t = j * T_k; \tag{5a}$$

$$k = 1, 2, \ldots, i; \tag{5b}$$

$$j = 1, 2, \ldots, \lceil T_i/T_k \rceil \tag{5c}$$

Combining Equation (3) and (5a, 5b, 5c) an algorithm can be implemented using Equation (4) as the checking condition. We shall refer this algorithm to as Time Demand Analysis (TDA). The algorithm for TDA is explained below.

We can divide the algorithm into three phases for better understanding:

1) Determining the Order of execution - The task-set is sorted as per increasing period. The sorted task sets are sequentially processed one by one. Every task is then subjected to TDA. The task, if any, at which the TDA gives the result as un-schedulable, is the threshold task. Since the task set is sorted, any task after that also will be un-schedulable.
2) Determining the Discrete time points - Schedulability test, as mentioned earlier need only be performed at certain time intervals. Using Equation (5) those time

Algorithm 1: TDA

```
 Input: task - set
 Output: schedulable or
 not
1 Sort the task-set;
2 repeat
3     Calculate time points from Equation (5);
4     Calculate w(t) from Equation (3);
5     if w(t) ≥ t then
6         return Not schedulable;
7         exit algorithm;
8     until all tasks in the sorted set have been processed;
9     return schedulable;
```

points are calculated for every task. These are the points where an instant of one of the task occurs and thus needs to be checked for schedule ability.

3) Computing the time demand function value - The value calculated from Equation (3) can be computed after all the time points are calculated. At every time point, this value is computed and continually summed over. The final sum is the required value that is compared to Equation (4). This value is then checked as per the Equation (4) which gives the decision whether schedulable or not.

B. Improved Time Demand Analysis

TDA can be further improved by using optimization techniques. The improved algorithm, termed as Improved Time Demand Analysis (ITDA) is given below. The equations required for the computation of the scheduability are also stated.

$$r = t/w(t) \tag{6}$$

$$r \geq 1 \tag{7}$$

$$m = k + [r * (n/MAX)] \tag{8}$$

where r is the ratio, m is the next task to be executed and MAX is the maximum value of the ratio.

C. Algorithm for ITDA

Algorithm 2: ITDA

```
 Input: task- set
 Output: schedulable or
Unschedulable
1  Sort the task-set;
2  k = 1;
3  repeat
4  Task considered k;
5      Calculate time points from Equation (5);
6      Calculate w(t) from Equation (3);
7          if w(t) ≥ t then
8              return Not-schedulable;
9          exit algorithm;
10     Calculate ratio r using Equation (6);
11     Calculate value m using
       Equation (8);
12     k = m;
13     until last task is not
       checked OR ratio ≥1;
```

```
14        return schedulable;
```

D. Parallel MITDA

The algorithm developed earlier can be modified to introduce parallelism for MITDA. The algorithm is described in steps given below.

1) Allocate the tasks to processors: The tasks are divided among the processors. This division is done by balancing the utilization using Utilization Balancing algorithm. This section is performed by the master thread. [23].
2) Create a team of threads: The number of threads created is equal to the number of processors present in the system. Each of the thread is given a processor to compute the Schedulability analysis for it.
3) Determine Schedulability of each processor: Each processor has a number of tasks allocated to it which are balanced in terms of total utilization. On each of these task-sets, ITDA is performed. The result obtained gives the Schedulability of each processor. Combining the results of the individual processors, the total Schedulability can be estimated. This function is carried out individually in each of the threads.

The algorithm is presented as Parallel MITDA. The environment is assumed to support at least 4 threads.

Algorithm 3: Parallel MITDA

```
Input: task- set, processors
Output: schedulable or
Unschedulable
 1 Sort the task-set;
 2 k = 1;
 3 Initialize utilization = 0 for each processor;
 4 repeat
 5      Find processor $P_k$ with min utilization;
 6      Assign next task $T_i$ to $P_k$;
 7      Update utilization of $P_k$+ = utilization of $T_k$;
 8 until all tasks have been allocated;
 9 Create a team of threads;
10 for each processor $P_k$ do
11      Allocate the processor to a thread;
12      Run ITDA() on each thread;
```

The parallel algorithm works better than serial implementation. Each of the processor's work is assigned to a separate thread which causes the total load to be divided.

E. Distributed Systems

A distributed system is a model in which components located on networked computers communicate and coordinate their actions by passing messages [15]. The components interact with each other in order to achieve a common goal. A distributed system may have a common goal, such as solving a large computational problem; the user then perceives the collection of autonomous processors as a unit [7]. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users. Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system. MPI is a specification for the developers and users of message passing libraries. By itself, it is not a library – but rather the specification of what such a library should be. MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process. Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be practical, portable, efficient and flexible MPICH is a freely available, portable implementation of MPI, a standard for message-passing for distributed-memory applications used in parallel computing [8]. It provides a flexible message passing the interface to carry out distributed computing. The performance of MPI has been analyzed properly in [9] where various performance measures have been proposed to be followed. This project uses MPICH as the API to implement distributed computing. The reason for using MPICH instead of other APIs is the ease of use and flexibility. Performance wise OpenMPI [5] is not that much different to that of MPICH. Various shortcomings of the MPICH are presented in the paper [10]. Combining the MPICH with multi-threading environment poses various challenges as accounted in [11]. To avoid such problems, we have not considered hybrid systems for the present project. In the next section we discuss the proposed algorithm.

III. LITERATURE SURVEY

In the paper [12], the authors present the implementation of MPICH2 over the Nemesis communication subsystem and the evaluation of its shared-memory performance. They describe design issues as well as some of the optimization techniques we employed. They conducted a performance evaluation over shared memory using micro benchmarks. The evaluation shows that MPICH2 Nemesis has very low communication overhead, making it suitable for smaller-grained applications. This paper shows that our motivation of pursuing distributed as an improvement over parallel is justified. The one-sided communication operations in MPI are in-tended to provide the convenience of directly accessing remote memory and the potential for higher performance than regular point-to-point communication. The work done in [13],[14] gives performance measurements with three MPI implementations (IBM MPI, Sun MPI, and LAM) which indicate, however, that one-sided communication can perform much worse than point-to-point communication if the associated synchronization calls are not implemented efficiently.

In this paper, they describe their efforts to minimize the overhead of synchronization in their implementation of one-sided communication in MPICH-2. They describe their optimizations for all three synchronization mechanisms defined in MPI: fence, post-start-complete-wait, and lock-unlock. Their performance results demonstrate that, for short messages, MPICH performs six times faster than LAM for fence synchronization and 50% faster for post-start-complete-wait synchronization, and it performs more than twice as fast as Sun MPI for all three synchronization methods.

IV. DISTRIBUTED MITDA

There are several sound reasons for applying Distributed Computing to problems. Reach Many Users: The users of a system are not in one place, however, system functionality implies a means of cooperating on me task or data.

Fault Tolerance - There is provision for failures and provides a backup system to take over. Load Sharing - This is the primary reason to use distributed computing. A huge increase in performance can be seen by using multiple nodes to increase performance.

A. Motivation

Out of the given points, parallel computation can achieve only Load Sharing. Due to the presence of shared memory as the communication mechanism, carrying out fault tolerance becomes highly difficult. Similarly the processors must be connected physically for shared memory architecture to work.

1) Shortcomings in Parallel Implementation: Although parallel execution shows improvement over serial execution, there are a few shortcomings that make it not fit for practical implementation. All the modern systems are hybrid systems. Parallel architecture can provide only up to a constrained amount of multi-programming. This prevents an algorithm to achieve enough performance gain even if the data supports fine grained parallelism. Shared memory architecture means that the logical processors or threads must be connected to each other via physical means. This restricts the hardware that can be used. In a multiprocessor environment, it should not be necessary that all the processors are physically connected to each other. They might be separated but connected by a network

Fault detection and correction is difficult in shared memory architecture. Failure of a processor generally means failure of the whole system as they are physically connected. This makes the system unreliable. Distributed system, as mentioned earlier, has advantage over parallel systems in this regard. Taking these motivations into account, we now propose a distributed algorithm which shows improvement over parallel algorithm.

B. Algorithm: Distributed MITDA

In a distributed system, the most important aspect is communication between nodes. It is of high essence that each of the nodes is assigned work properly. In this project, we are not concerning ourselves with the creation of a distributed cluster system. We consider a functional 4 node cluster on which we carry out the implementation. The algorithm is explained step wise below.

1) Allocate the tasks to processors: The tasks are divided among the processors. This division is done by balancing the utilization using Utilization Balancing Algorithm. This section is performed by the master thread. [23].
2) Send each node the allocation of a processor: With the help of message passing, each node is passed the allocation of a processor. The node then computes and communicates back the Schedulability result
3) Determine Schedulability of each processor: Each processor has a number of tasks allocated to it which are balanced in terms of total utilization. On each of these task-sets, ITDA is performed. The result obtained gives the Schedulability of each processor. Combining the results of the individual processors, the total Schedulability can be estimated. This function is carried out individually in each of the threads.

Now we present the algorithm Distributed MITDA. The input to this algorithm is number of processors, task-set and a host file that contains the IP address of the nodes in the cluster.

Algorithm 5: Distributed MITDA

```
Input: task -set, processors, host file
Output: schedulable or
Unschedulable
1 Sort the task-set;
2 k = 1;
3 Initialize utilization = 0 for each processor;
4 repeat
5     Find processor $P_k$ with min utilization;
6     Assign next task $T_i$ to $P_k$;
7     Update utilization of $P_k$+ = utilization of $T_k$;
8 until all tasks have been allocated;
9 Initialize each node $n_k$ using IP address from hostfile;
10 Invoke MPI function MPIinit;
11 Communicate with each node $n_k$ using MPIsend;;
12 for For each node $n_k$ do
13 Pass the allocation of the processor to the node;
14 Run ITDA() on the node;
15 Synchronize the nodes using MPIbarrier;
16 Receive result from the node;
```

In the algorithm, we first sort the task-set and then allocate each task to a processor. This is done by a single node. After allocation to each processor is completed, communication with nodes is initiated. Each processor is communicated with the allocation of a single processor in step 12. Then each of the nodes computes the Schedulability of the assigned processor. The result of the Schedulability test is sent back to the central node. The algorithm ends when all the nodes have completed their execution.

C. Implementation

For implementing this algorithm, a distributed architecture is considered already to be in a functional state. The IP address of all the nodes is maintained in a host file which is used during the compilation time to link the program to each node. Further to communicate among nodes, MPICH API is being used. MPICH is a high performance and widely portable implementation of the Message Passing Interface (MPI) standard. The primary reason for using MPICH is its flexibility and scalability. The number of processors used and the number of threads used is equal to 4 for parallel and the number of nodes are equal to 4. Each of the node is a standalone processor with no shared memory with the other nodes. Thus every form of communication has to be done using MPI. The communication was carried out using MPIsend() and MPIreceive() methods as defined by MPICH.

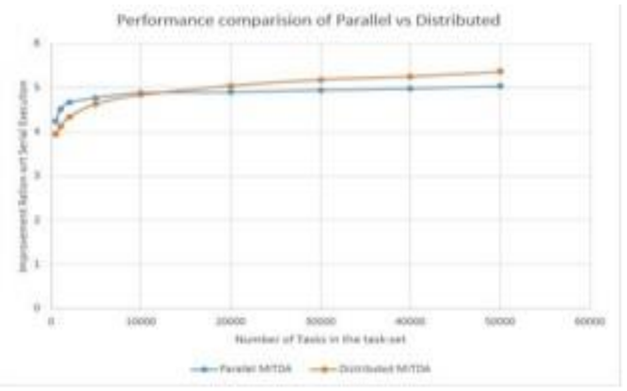

Fig 1. Comparison of Parallel and Distributed MIDTA wrt Serial MITDA

D. Results

As per the progress in implementation, the results obtained are shown in the table. The table shows the number of iterations taken and the CPU Time taken for Parallel and distributed execution of MITDA. The figure shows the comparison of parallel and distributed both with serial. This shows a relative increase in performance. This is expected as when the number of tasks is smaller the communication overhead was significant. In parallel computation, the shared memory took care of this extra communication overhead. The threads made it easier for the shared memory access. In distributed computing, the allocation of each processor must be communicated. When we increase the number of tasks the gain by using a dedicated node pays off.

TABLE I. COMPARISON BETWEEN PARALLEL AND DISTRIBUTED MITDA TAKING NUMBER OF PROCESSORS =
4, NUMBER OF THREADS = 4, NUMBER OF NODES = 4

No of Tasks	Ratio of Parallel to Serial	Ratio of Distributed to Serial
500	4.25	3.954
1000	4.52	4.12
2000	4.67	4.34
5000	4.78	4.64
10000	4.89	4.84
20000	4.91	5.05
30000	4.95	5.19
40000	4.99	5.26
50000	5.04	5.37

V. CONCLUSION

In this paper we saw the Schedulability test for RMA - ITDA and MITDA. We explored the possibility of extending the MITDA to parallel systems and taking advantage of the multiple number of processors present. The proposed Parallel MITDA algorithm is shown to give a high performance ratio than serial. This result shows that in a multiprocessor environment, the serial execution can be replaced by parallel execution.

REFERENCES

1. C. L. Liu and J.W. Layland ”Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment, Journal of the Assn of Computing Machinery (ACM) 20, 1, 40-61 January, (1973)
2. Lehoczky, John, Lui Sha, and Ye Ding. ”The rate monotonic scheduling algorithm: Exact characterization and average case behavior.” Real Time Systems Symposium (1989)
3. Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin, On Task Schedulability in Real-Time Control System, Proceedings of 17th Real-Time Systems Symposium, pp. 13-21, December, (1996)
4. T. W. Kuo, Y. H. Liu, and K. J. Lin, Efficient On-Line Schedulability Tests for Priority Driven Real-Time Systems, Proceedings of the IEEE Symposium on Real- Time Technology and Applications, Washington D.C., USA, June (2000).
5. Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. Using OpenMP: portable shared memory parallel programming. MIT press, Vol. 10, (2008).
6. Audsley, Neil, et al. ”Applying new scheduling theory to static priority pre-emptive scheduling.” Software Engineering Journal 8.5 pp. 284-292. (1993)
7. Birman, Kenneth P., and Robbert van Renesse, eds. Reliable distributed computing with the Isis toolkit. IEEE Computer society press, Vol 10, (1994).
8. W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, (1999).
9. J. L. Trff, W. Gropp, and R. Thakur, Self-Consistent MPI Performance Requirements, in Proc. of the 14th European PVM/MPI Users Group Meeting (Euro PVM/MPI 2007), pp. 36-45, September (2007),
10. R. Thakur and W. Gropp, Open Issues in MPI Implementation, in Proc. of the 12th Asia-Pacific Computer Systems Architecture Conference, pp. 327-338. August (2007)
11. W. Gropp and R. Thakur, Issues in Developing a Thread-Safe MPI Implementation, in Proc. of the 13th European PVM/MPI Users Group Meeting (Euro PVM/MPI 2006), pp. 12-21, September (2006)
12. D. Buntinas, G. Mercier and W. Gropp, Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem, Parallel Com-puting, Volume 33, Issue 9, pp. 634-644. September (2007)
13. R. Thakur, W. Gropp, and B. Toonen, Minimizing Synchronization Overhead in the Implementation of MPI One-Sided Communication, in Proc. of the 11th European PVM/MPI Users Group Meeting (Euro PVM/MPI 2004), Recent Advances in Parallel Virtual Machine and Message Passing Interface,
14. Lecture Notes in Computer Science, LNCS 3241, Springer, September (2004)
15. Garibay-Martnez, Ricardo, et al. ”On the scheduling of fork-join parallel/distributed real-time tasks.” Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on. IEEE, (2014)

16. Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair. Distributed Systems: Concepts and Design (5th Edition), (2011)
17. Bini, Enrico, and Giorgio Buttazzo. ”A hyperbolic bound for the rate monotonic algorithm.” Real-Time Systems, 13th Euromicro Conference on, 2001.. IEEE, (2001).
18. Allah, Nasro Min, and S. Islam and Wang YongJi. ”Enhanced Time Demand Analysis.” World Applied Sciences Journal 9.1 (2007)
19. Manabe, Yoshifumi, and Shigemi Aoyagi. ”A feasibility decision algo-rithm for rate monotonic and deadline monotonic scheduling.” Real-Time Systems 14.2 pp. 171-181 (1998)
20. Bini, Enrico, and Giorgio C. Buttazzo, ”Schedulability analysis of periodic fixed priority systems.” IEEE Transactions on Computers 53.11 pp. 1462-1473, (2004).
21. Bini, Enrico, and Giorgio C. Buttazzo. ”The space of rate monotonic schedulability.” In Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE, pp. 169-178. IEEE, (2002).
22. Thomas H.. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. ”Introduction to algorithms”. Vol. 6. Cambridge: MIT press, (2001).
23. Enrico Bini and Giorgio C. Buttazzo. ”Measuring the Performance of Schedulability Tests”, Springer, Real-Time Systems, Volume 30, Issue 1, 122 154, May (2005)
24. Rajib Mall. ”Real Time System: Theory and Practice” 3rd Edition, Pearson Education pp 390-395, (1986)
25. Lalatendu Behera and Durga Prasad Mohapatra, ”Schedulability Analy-sis of Task Scheduling in Multiprocessor Real-Time Systems Using EDF Algorithm”, International Conference on Computer Communication and Informatics, Jan. (2012)
26. F. Zhang and A. Burns. ”Schedulability Analysis for Real-Time Systems with EDF scheduling” , IEEE Transaction on computers, vol. 58, no. 9, pp. 1250-1258, September (2009)
27. Sjodin, Mikael, and Hans Hansson. ”Improved response-time analysis calculations.” Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE, (1998).
28. Joseph, Mathai, and Paritosh Pandya. ”Finding response times in a real-time system.” The Computer Journal 29.5, pp. 390-395, (1986)

AUTHORS PROFILE

Leena Das is currently pursuing PhD under KIIT University, Bhubaneswar,Odisha.She is working as Assitantant Professor in the department of SCE . She has done her Ms from BITs ,Pilani. She is life member of Indian Science Congress and a member of IET.Her current research area is Real-Time System and Software Engineering.

Dr Durga Prasad Mohapatra completed his PhD from Indian Institute of Technology, Kharagpur in 2005 and has since been a professor in National Institute of Technology. He is currently the Head of Department, Computer Science and Engineering. He is a member of IEEE technical society. His current research area is Software Engineering, Service Oriented Architecture ,Real-Time system.