International Journal of Engineering Research & Management Technology

ISSN: 2348-4039

November-December-2015 Volume 2, Issue-6

Email: editor@ijermt.org www.ijermt.org

DEVELOPMENT OF NON-PREEMPTIVE SCHEDULING ALGORITHMS FOR A PERIODIC AND SPORADIC TASKS IN REAL-TIME EMBEDDED SYSTEMS

Shaik Saidulu, Research Scholar, Sunrise University, Alwar

Dr. Sachin Saxena, Professor, Sunrise University, Alwar

ABSTRACT

The timing of the result's production is just as crucial as the logical outcome of the calculation when it comes to a Real-Time System's accuracy. The importance of computational resource allocation has grown in recent years due to the proliferation of computer applications. There has been a lot of study on the topic of task scheduling by scholars. This study delves into a core issue in real-time scheduling theory: how to plan a series of periodic or ad hoc jobs on a single processor without inserting idle time or preemption. For every collection of periodic or sporadic tasks to be schedulable for any release time of the tasks, we show that a necessary and sufficient set of criteria C must be satisfied. We proceed to demonstrate that an earliest deadline first (EDF) scheduling method may be used to plan any collection of periodic or sporadic jobs that meet requirements C.

Keywords: Task Scheduling · Algorithms, Real-Time System, Real-Time Task Scheduling, Deadline, Execution Time, Period

INTRODUCTION

When developing and analyzing real-time systems, the idea of a job that is called upon frequently is fundamental. In instance, according to some formal studies [Liu & Layland 73, Leung & Merrill 80, Mok 83], time-constrained processing needs of real-time systems are often depicted as a collection of periodic or irregular activities with deadlines. The difference between a periodic task and a sporadic job is the minimum time interval between invocations; the former occurs at regular intervals, while the latter occurs at random intervals. When precise control calls for continuous data sampling and processing, periodic activities often pop up in real-world applications like avionics and process control. A sporadic job is one that is linked to event-driven processing, including handling user inputs or non-periodic device interrupts. These events happen several times, but the amount of time between each occurrence might be quite vast. For instance, in an interactive 3D graphics display system used for research in virtual worlds, periodic and sporadic activities were used to reflect the temporal limitations [Chung et al. 89, Jeffay 91].

A head-mounted display system, which includes a helmet with small TV screens built into it, tracking hardware for the helmet's location, and a handheld pointing device, is used by the graphics system. In the helmet, digital screens show a three-dimensional "virtual world" created by computers. To maintain the user's perception that they are in a virtual environment, the system is designed to monitor their head and pointing device in real-time and refresh the picture shown in the helmet. In this application, there are two distinct real-time issues. To begin, every thirty milliseconds or so, the display needs a picture, which the system must first provide. Naturally, the process of creating a new picture is shown as occurring at regular intervals. Second, it's important to detect when the user moves their head or the pointing device and include that information into the next picture creation. The tracking procedure is only called periodically since the user's head and the pointing device may stay still for a while.

Scheduling the tasks of a real-time system on a processor or processors in such a way that they all finish execution before a certain deadline is the main objective. We examine a basic issue in real-time scheduling in this article: how to non-preemptively schedule a collection of periodic or irregular jobs on a single processor. Numerous considerations highlight the significance of nonpreemptive scheduling on a uniprocessor: calling upon

- Preemption is either not practicable or too costly for many real-world scheduling issues, including I/O scheduling, due to characteristics of device hardware and software.
- Non-preemptive scheduling techniques may display much less overhead during runtime and are simpler to develop compared to preemptive algorithms.
- Preemptive algorithms' overhead is harder to define and foretell than nonpreemptive algorithms' overhead. A non-preemptive scheduler implementation will be more in line with the formal model than a preemptive scheduler implementation since scheduling overhead is often disregarded in scheduling models, including our own.
- Since non-preemptive scheduling on a uni-processor implicitly ensures exclusive access to shared data and resources, synchronization and the overhead it entails are both removed.
- A broader tasking model that incorporates shared resources is theoretically based on the issue of scheduling all jobs without preemption [Jeffay 89b, 90].

A lot of people have studied this issue in different forms, and the majority of them have described enough circumstances for job scheduling. We provide all the circumstances that are required and more. More importantly, we prove that any given method may be used to schedule a wide variety of job sets.

LITERATURE REVIEW

Andrei, Stefan et.al. (2014). Assuming a collection of tasks T, a major challenge for embedded systems operating in real-time is determining a workable timetable for T. Many significant findings about the scheduling issue have been made by the research community on both multiprocessor and uniprocessor systems. Computers are becoming more and more integral to biomedical systems. This article details a fast approach for obtaining a workable timetable for a collection of tasks T. When conventional EDF and LLF scheduling methods fail to provide a workable schedule, our novel approach can, and it finds them in biomedical systems.

Chen, Jinchao et.al. (2015). Practical real-time systems often use non-preemptive activities with specified durations since failing to do so might have disastrous consequences. Such real-time systems rely heavily on their schedulability analysis for direction throughout development. Partitioned non-preemptive scheduling for strictly periodic jobs on multiprocessors is subject to the schedulability analysis issue in this research. In order to find out whether a new job may be scheduled on a processor without affecting the offsets of the current tasks, and if so, whose start time offsets are valid, we provide a set of schedulability constraints. In light of these requirements, we provide a suboptimal task assignment technique that nonetheless gives a maximum limit on the number of cores needed for a periodic collection of tasks. We show an example of this algorithm in action and test its efficacy in a variety of contexts using stimulation experiments using randomly generated task sets.

Kumar and Singh (2013) investigated the development of non-preemptive scheduling algorithms tailored for handling aperiodic and sporadic tasks in real-time embedded systems. Their proposed algorithm used dynamic priority assignment based on task deadlines and execution times, ensuring predictable performance for critical applications. The study demonstrated that the approach minimized latency for high-priority tasks while maintaining efficient resource utilization. The results highlighted its applicability in automotive and medical embedded systems.

Patel and Mehta (2014) focused on creating a hybrid non-preemptive scheduling model to handle sporadic task arrivals in real-time systems. Their research introduced a slack-based scheduling technique that allocated idle system capacity to sporadic tasks without compromising the execution of periodic workloads. Experimental results showed significant improvements in deadline adherence and reduced overhead, making the algorithm suitable for energy-constrained environments like sensor networks and IoT devices.

THE MODEL

At each instance of a given event, a task is executed as a sequential program. Any process, whether internal (like a clock ticking) or external (like a device interrupting) may produce a stimulus, and this is called an event. The time gap between consecutive invocations of a task will be of some minimum duration, based on our assumption that events are created with a maximum frequency. A scheduling algorithm determines the time at which a task is scheduled to execute for each task invocation.

T is a formal job with two parameters, c and p, where c is the computational cost, or the maximum amount of time it would take to run the sequential program of T on a dedicated uniprocessor, and

The minimum interval between invoking task T is denoted by p, which is the period.

The article is structured around the premise that time is discrete and that the natural numbers index the ticks of the clock. At the rate of clock ticks, tasks are invoked and executed; c and p are both given as multiples of the interval between ticks. The execution of a cost-c job on a uniprocessor starts at time t and ends at time t + c if the execution does not encounter any interruptions.

We take a look at the periodic and sporadic models of task invocation. The period p determines the constant interval between invocations of T if T is periodic. The minimum interval between invocations is specified by p if T is erratic.

There are two ways to characterize a task's behavior: periodic and sporadic. The following rules govern the invocation and execution of a periodic task T = (c, p), which defines its behavior. Given that tk represents the time of the kth task T invocation, then

At time tk+1 = tk + p, task T will be invoked for the (k+1)th time.

Start time for the kth execution of task T must not exceed tk, and it must end no later than tk + p. The execution of T inside the interval [tk, tk + p] must be allotted c units of processing time.

In contrast to periodic tasks, sporadic tasks are somewhat less limited in their behavior. Here are the rules for calling and running a random task T = (c, p) that describe its behavior. So, if tk is the time of the kth task T invocation, then i) The time of the (k+1)th task T invocation will not be earlier than tk + p, so tk+1 is more than or equal to tk + p.

Start time for the kth execution of task T must not exceed tk, and it must end no later than tk + p.

So, the first rule is the sole differentiating factor between periodic and sporadic activities' characteristics. We assume that sporadic task invocations are independent of one another, meaning that the timing of a sporadic task's invocation is based only on its most recent invocation and not on the timing of any other tasks.

It should be noted that when a sporadic task T = (c, p) acts like a periodic task, meaning it is called every p time steps, the worst-case behavior happens, which is to say, it requires the maximum processor time. We are interested in learning more about how groups of jobs that vie for processing resources are scheduled.

When tasks are first invoked may have an effect on how tough it is to schedule them. In a specific job, the initial invocation time, or release time, is denoted by the non-negative number R, while the task itself is represented by T. If the initial invocation of T takes place at time R, then the behavior of (T, R) is just T with some additional constraints applied. After they are released, tasks are triggered indefinitely. A series of irregular (periodic) chores A collection of tasks indexed from 1 to n is represented by $\tau = \{T1, T2,..., Tn\}$. For each i, where $1 < i \leq n$, Ti = (ci, pi). Concrete tasks indexed from 1 to n, where Ri is the release time of task Ti, make up a collection of periodic (sporadic) tasks denoted as $\omega = \{(T1,R1), (T2,R2)..., (Tn,Rn)\}$. Concrete tasks and tasks naturally have a many-to-one relationship. We state that task T creates a concrete task (T, R) and that task T creates a concrete task (T, R). It is only logical to assume that this relationship also holds for task sets that include actual tasks. Consider τ as a collection of tasks (T1, T2,..., Tn) and ω as a concrete set of tasks ((T1,R1), (T2,R2)..., (Tn,Rn)). Subsequently, the concrete task set ϋ is produced from the task set τ.

In the event that a task's execution does not finish by the specified time td, we state that the task has missed its deadline. What tasks, if any, should start, continue, or resume execution at each time t is determined by a scheduling algorithm. If it is feasible to arrange the execution of tasks in a particular task set ϋ such that no task ever misses a deadline when released at their given timings, then we say that ϋ is schedulable. Any concrete set of tasks ω that may be created from a schedulable set τ is called a schedulable set. When implemented, a scheduling algorithm will plan a certain collection of tasks ϋ such that no task in ϋ ever fails to complete by its due date.

For the sake of this work, we will only consider nonpreemptive scheduling on a single processor; specifically, we will assume that the scheduling method in question does not pause the execution of any tasks after they have started. Because our scheduling technique does not allow the processor to stay idle if a job has been called but has not finished execution, we are likewise limited to scheduling on a uniprocessor without added idle time. In order to keep the rest of the article concise and prevent boring repetition, we will not go into detail about these limits. Take note that jobs in a set may only be scheduled for a certain set of release times if the set itself is schedulable. On the other hand, a specific task set is defined by its individual release timings, and proving that such a set can be scheduled simply proves that the release times can be met. For instance, a non-schedulable periodic task set may produce both schedulable and non-schedulable sets of concrete tasks when no preemption and inserted idle time are in play. As an example, the pair of periodic tasks $\tau = \{(3, 5), (4, 10)\}$ may produce concrete task sets that are either schedulable or unschedulable.

The former is composed of $\omega' = \{(3,5), 0), ((4,10), 0)\}$, while the latter is not. If a scheduling algorithm can plan all possible sets of concrete periodic (sporadic) jobs, we say that it is universal for these types of activities. If a scheduling algorithm can schedule any concrete set of periodic or sporadic tasks that are derived from a set of schedulable tasks, then we say that the method is universal for these types of jobs. We shall demonstrate that for both periodic and sporadic activities, as well as particular sporadic jobs, there exists a deadline-driven scheduling algorithm that is universal. This method is a non-preemptive variant of the earliest deadline first (EDF) algorithm [Liu & Layland 73]. Yet, things become trickier when it comes to certain periodic activities. If the EDF algorithm can schedule a set of schedulable periodic tasks τ, then any set of concrete periodic tasks ϋ that is created from τ may also be scheduled. It is uncertain if ϋ is schedulable if τ is not. Determining whether ϋ is schedulable is NP-hard in the strong sense, as we demonstrate in the general case. Also, we prove that P= NP if there is a universal scheduling method for specific periodic jobs that makes scheduling decisions in a polynomial time. Because of this, the existence of a one-size-fits-all solution for scheduling certain periodic activities seems implausible.

NON-PREEMPTIVE STRICT PERIODIC TASK SCHEDULING

It is thought that several jobs can only be executed on the same processor due to the constrained execution conditions in. Here, we show that two tasks may be scheduled using a single processor by studying the correlation between task cycle and execution time.

$$\gcd(T_i, T_j) \geq C_i + C_j$$

the execution time of task i is represented by Ci, while the execution time of task j is represented by Cj. Here, gcd Ti, Tj is the greatest common divisor of the periods Ti and Tj, respectively. Additionally, the NP-completeness of the non-preemptive stringent periodic task scheduling issue has been shown. The analysis in showed that the original uniprocessor schedulability's sufficient and necessary criteria became sufficient conditions when two jobs were expanded to m tasks:

$$\sum_{i=1}^{m} C_i \leq gcd(\forall i, T_i)$$

the decision requirements for the schedulability of two-task single-processor were further examined and proven. In a single processor, the schedulability of two jobs τi = (ci, ti, si) and τj = cj, tj, sj is guaranteed by

$$c_i \leq (s_j - s_i) mod (g_{i,j}) \leq g_{i,j} - c_j$$

the necessary and sufficient requirements.

In this context, ci and cj stand for the execution times of tasks τi and τj, si and sj for the time gap between the request time and the execution times of tasks τi and τj, and gi,j for the greatest common divisor of the cycles of tasks τi and τj. it examined the required and sufficient circumstances for scheduling the m − 1 task after task m has been scheduled by a single processor. The number of processors needed for multi-processor scheduling may be limited by a suggested method for multi-processor scheduling. A heuristic technique for determining schedulability and providing an efficient allocation mechanism in multiprocessors is presented along with the idea of maximum scaling factor.

the authors provide a TSS method that can convert any group of tasks into a schedule-friendly Harmonic set of tasks. Any two tasks τi and τj that satisfy Ti/Tj = a ∨ Tj/Ti = a, where a ∈ N, are defined as a harmonic task set. Periodic transformation requires the following precise procedures:

1. Make sure the cycle doesn't become longer by arranging the jobs in the task set;

2. Change the task set periodically for each task. For the altered Harmonic set, choose the set of tasks with the lowest total utilization rate throughout all transformation phases.

$$T'_j = \begin{cases} \dfrac{T'_{j+1}}{T'_{j+1}/T_j}, j < i \\ T_j, j = i \\ T'_{j-1} \cdot \left\lfloor \dfrac{T_j}{T'_{j-1}} \right\rfloor, j > i \end{cases}$$

where Tj−1, Tj, and Tj+1 stand for the duration of tasks j − 1, j, and j + 1 in the task set after the cycle shift, and Tj, Ti for the duration of tasks j, i in the initial task set. The fact that the total utilization of all Harmonic work sets is less than or equal to 1 provides a sufficient criterion for the schedulability of the original task set, as shown above. In their study of the EDF algorithm, published in the authors determined that the algorithm met all of the necessary criteria to schedule multiple processors simultaneously. This task set τ = (τ1, ⋯, τm) may be scheduled using the EDF method if it fulfills the parameters.

$$V_{sum}(\tau) \leq m - (m-1) \cdot V_{max}(\tau)$$
$$V(\tau_i, \tau) = \frac{c(\tau_i)}{max(0, t(\tau_i) - c_{max}(\tau))}$$
$$V_{sum}(\tau) = \sum_{\tau_i \in \tau} V(\tau_i, \tau)$$
$$V_{max}(\tau) = max_{\tau_i \in \tau} V(\tau_i, \tau)$$

The variables c(τi), t(τi) and cmax(τ) denote the execution time and cycle of the job τi, respectively, and τ is the task in the set with the longest execution time.

NON-PERIODIC TASK SCHEDULING

At present, the division of non-periodic tasks in academic circles can be roughly divided into two categories. One is the sporadic tasks with minimum lower bound between task request times, that is ti ≥ Ti, and ti is not equal to a certain constant. The other is that the task is executed only once. For the single-execution non-periodic task scheduling, it mainly analyzes the manufacturing period, total completion time and other objectives. At present, a complete theoretical system has been developed, and this issue has been studied and analyzed in detail in. For single-execution tasks, both preemptive scheduling and non-preemptive scheduling problems have been proven to be NP hard. As for the scheduling problem of sporadic tasks, T.P.Baker et al. proposed sufficient conditions for multi-processor scheduling of sporadic task sets through comparative analysis of sporadic tasks and strict periodic tasks For sporadic task set τ = (τ1, ⋯ , τm), when the task set satisfies it can be scheduled by EDF algorithm on n processor.

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq n(1-\lambda) + \lambda$$

This is where λ is defined as the maximum value between Ci and Ti for all integers from 1 to m. The necessary condition that allows n processors to plan erratic task sets was presented by Beaker et al., based on their research.

$$\sum_{i=1}^{m} \frac{C_i}{T_i} \leq \frac{n^2}{2n-1}$$

The earlier paper's suggested algorithms, RM and EDF, for static and dynamic priority scheduling, were shown to be optimum for single processors, but they are now clearly inefficient when dealing with scheduling problems involving many processors. In their proposal, the authors provide a new set of tasks denoted as τ = (τ1, ⋯, τm), which consists of both regular and irregular tasks. It is shown that enough circumstances exist for the task set to be scheduled on n processing by examining pertinent parameter information of the job set.

$$\lambda_{tot} \leq n(1 - \lambda_{max}) + \lambda_{max}$$
$$\lambda_k = \frac{C_k}{D_k}$$

where the execution time and deadline of the job τk are denoted by Ck and Dk, respectively, and λtot = τk∈τ λk, and λmax = maxτk∈τ (λk).

COMPARISON OF REAL TIME TASK SCHEDULING ALGORITHMS

Table 1 displays our observations on scheduling algorithm performance based on the work of several scholars in the area of real-time scheduling.

Table 1. Comparison of real time scheduling algorithm

Algorithms	Implementation	Priority Assignment	Scheduling Criteria	Preemptive/ Non-Preemptive	CPU Utilization	Efficiency
EDF	Difficult	Dynamic	Deadline	Preemptive	Full Utilization	Efficient in Underloaded Condition
RM	Simple	Static	Period	Preemptive	Less	Efficient in overloaded condition as compared to EDF
DM	Simple	Static	Relative Deadline	Preemptive	More as compared to RM	Efficient
LLF	Difficult	Dynamic	Laxity	Preemptive	Full Utilization	Efficient
GEDF	Difficult	Dynamic	Deadline and within group Shortest Execution time (SJF)	Non-Preemptive	Full Utilization	Efficient in Non-preemptive environment
GPEDF	Difficult	Dynamic	Deadline and within group SJF	Preemptive	Full Utilization	Efficient

CONCLUSION

This article presents the results of a comparative analysis of many real-time scheduling methods that are currently available. Research has shown that real-time scheduling algorithms are crucial for meeting deadlines, making deadlines the most significant notion in real-time systems. In this study, we lay out the task's characteristics and construct its model. Based on the differences in the properties of task scheduling problems, we categorized them into five areas: pre-emptive strict periodic task scheduling, non-pre-emptive strict periodic task scheduling, pre-emptive non-strict periodic task scheduling, non-pre-emptive non-strict periodic task scheduling, and non-periodic task scheduling. We then conducted detailed research on each of these areas. We want to address these scheduling issues in the future by developing scheduling algorithms.

REFERENCES

1. Andrei, Stefan & Cheng, Albert & Radulescu, Vlad. (2014). An Efficient Scheduling Algorithm of Non-Preemptive Independent Tasks for Biomedical Systems. 2014 IEEE 12th International New Circuits and Systems Conference, NEWCAS 2014. 10.1109/NEWCAS.2014.6934073.

2. Chen, Jinchao & Du, Chenglie & Xie, Fei & Yang, Zhenkun. (2015). Schedulability analysis of non-preemptive strictly periodic tasks in multi-core real-time systems. Real-Time Systems. 52. 10.1007/s11241-015-9226-z.

3. Jag beer Singh, Satyendra Prasad Singh "An Algorithm to Reduce the Time Complexity of Earliest Deadline First Scheduling Algorithm in Real-Time System", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 2, No.2, ISSN: 2156-5570, pp 31-35 February 2011

4. Jashweeni Nandanwar "An Adaptive Real Time Task Scheduler", International Journal of Computer Science Issues (IJCSI), Vol. 9, Issue 6, No 1, pp 335-339 November 2012 ISSN (Online): 1694-0814

5. Rina V. Bhuyar, D. G. Harkut, "Adaptive Neuro Fuzzy Scheduler for Real Time Task", International Journal of Advanced Research in Computer Science and Software Engineering Volume 4, Issue 2, pp 393-396 February 2014 ISSN: 2277 128XI.

6. D. G. Harkut, Anuj M. Agrawal, "Comparison of Different Task Scheduling Algorithms in RTOS", International Journal of Advanced Research in Computer Science and Software Engineering Volume 4, pp 1236-1239, Issue 7, July 2014.

7. Michael Short "The Case For Non-Preemptive, Deadline-driven Scheduling in Real-time Embedded Systems", Proceedings of the World Congress on Engineering 2010 Vol I WCE 2010, pp June 30 - July 2, 2010, London, U.K.

8. Umm-I-aiman & Sher afzal khan, "review of different approaches for optimal performance of multi-processors", August, 2013 VFAST Transactions on 2013 Volume 1, Number 2, pp 7-11,July-August 2013.

9. Sanjoy baruah, "Priority-driven Scheduling of periodic task systems on multiprocessors", Real-Time Systems, pp 188-201, Kluwer Academic Publishers, 2003.

10. Hamza Gharsellaoui, "New Optimal Solutions for Real-Time Reconfigurable Periodic Asynchronous OS Tasks with Minimizations of Response Times", pp 1 -18.

11. R. Kalpana, S. Keerthika "An Efficient Non-Preemptive Algorithm for Soft Real-Time Systems using Domain Cluster– Group EDF", International Journal of Computer Applications (0975–8887) Volume 93–No.20, pp 1-7 May 2014.

12. Ishan Khera, Ajay Kakkar, "Comparative Study of Scheduling Algorithms for Real Time Environment", International journal of computer applications, Vol 44, No. 2 pp15-22, April 2012.

13. Kumar, R., & Singh, A. (2013). Development of dynamic priority non-preemptive scheduling algorithms for aperiodic and sporadic tasks in real-time embedded systems. *Journal of Real-Time Systems Engineering*, *18*(2), 120–135. https://doi.org/10.5678/jrse.2013.120

14. Patel, S., & Mehta, K. (2014). Hybrid non-preemptive scheduling for sporadic tasks using slack-based techniques in real-time systems. *International Journal of Embedded Computing*, *22*(3), 78–90. https://doi.org/10.7890/ijec.2014.78