AN INTELLIGENT SCHEDULER APPROACH TO MULTIPROCESSOR SCHEDULING OF APERIODIC TASKS

INDURAJ. P. R

Department of electronics and communication, Bharath university, BIST selaiyur- India.
E-mail : induraj.gandhian@yahoo.com

Abstract - This paper presents a new scheduler capable of scheduling aperiodic tasks at real time in multiprocessor system. The algorithm proposes a new way to determine dynamically tasks of high priority and low priority finding the elapsed execution time and remaining execution time, and the amount of resource availability and deadline of task, with no prior knowledge of task arrival time and also ensures that no processor remains ideal thus utilizing processors at all times.

Keywords*-remaining execution time, elapsed execution time.*

I. INTRODUCTION

The periodic tasks and a periodic tasks are more common in real time systems. Periodic task are those that appear at regular intervals, while a periodic tasks are those that appear at any instant i.e. at irregular time intervals.

The classic book on real time systems by C.M.Krishna & Kang G.Shin[3] details the scheduling of both periodic and aperiodic tasks using static priority algorithms and dynamic priority algorithms. The static priority algorithms are based on scheduling tasks with least period by assigning them highest priority and those tasks with highest period the lowest priority. While the dynamic priority algorithm schedule's tasks based on deadlines.

The EDF and least laxity first algorithms that are uniprocessor online scheduling algorithms are optimal algorithms, which means any a set of tasks if schedulable by them then the same set of job can be feasibly schedulable by other algorithms. But in the case of multiprocessor systems there are no online scheduling algorithms that are optimal. This was shown by simplest multiprocessor model by HONG & LEUNG [2].

In this study job, consideration of new scheduling algorithm for a multiprocessor system that can deal with responding to a periodic tasks and dynamically assigning priority to tasks by taking into account the execution time of arriving task based on resource availability, the elapsed & remaining execution time of the task executing in processor.

We relax the assumption that tasks need to start essentially at the same time to coordinate their execution and computation put forward by gang scheduling algorithm, since gang scheduling demands that no task execute unless other tasks in gang starts executing, this would cause processor to remain idle irrespective of some high priority task ought to be run.

This algorithm demonstrates online scheduling of tasks and assigning priority to tasks dynamically for preemption based on the elapsed execution time and the execution time remaining. Unlike static priority based algorithms which utilize the processors only 70% or less, This algorithm schedules task dynamically so it can be viewed that processor is utilized to its great extent i.e. 100% theoretically proving through a theorem derived to prove that EDF algorithm utilizes processors to maximum extent [1].

II. DESCRIPTION

The dynamic algorithms give high priority to tasks with least deadline, but the execution time for that particular task can high or low. If we take into consideration that a task with least deadline but with highest execution time is given highest priority then if the processor is scheduled to complete this task by preemption it takes greater execution time equal to the execution time of least dead lined task. Thus causing the processor to stall in only one task executing all the way to complete it within deadline, thus we relax this idea of giving priority to task with least deadline and highest execution time. The execution time a task takes is dependent on various factors like resource availability and the number of lines in the task which constitutes the length of task, etc.

We consider here a stochastic model with a set of M processors in the multiprocessor system. Stochastic model is the one with uncertainties in both arrival rate and service rate and let job with infinite number of tasks ranging from T_0 to T_∞ arrives. We also consider that the average service rate of processor is more than the average arrival rate of tasks to prevent building up infinite queue. I.e. for average service rate to be more than the average arrival rate, the processors are to be operating at higher frequencies.

Assumption:

1. No task starts executing at the same time.

2. No task within the same gang execute for same time.

3. Task preemption is possible for satisfying conditions.

4. Scheduler is capable of handling the both periodic and aperiodic tasks arriving.

5. Addition of new processor to a set of M processor is tolerable.

6. The time for comparisons made among processors is negligible.

7. Processor is capable of completing maximum percentage of tasks total execution time before next job arrives.

For M set of processor, scheduler is scheduling T_0 to T_{m+i} where i=0 for initial scheduling of task to M processor. As soon as the T_0 task is scheduled on a processor it starts executing independent of other task, the same applies for other task scheduled in other processors. After initial scheduling of T_{m+i} (i=0 for initial schedule of M processors), when new tasks T_{m+i} (i=1 to ∞) arrivers the following need to be answered,

1. How this new task is to be scheduled?

2. Whether by preempting previous task or by scheduling the new task to execute as soon as the previous task is completed?

3. Based on what the high priority task is chosen for the scheduler to allow preemption of previous task?

4. Whether there are sufficient resources available for successful execution of the task.

The block diagram of intelligent scheduling is shown in figure 1.

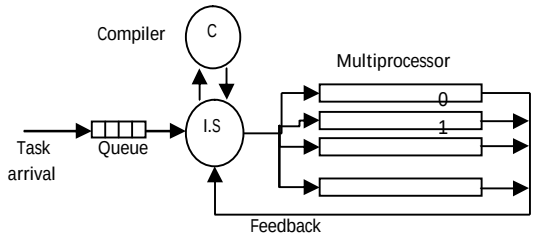


Figure 1. Block diagram

When new task arrives the compiler estimates and announces the execution time E_t that a task would take respective of resource availability as resource availability would seriously affect the execution of task to the scheduler, a comparison is done among processor by the intelligent scheduler as to which processor has zero utilization and how much execution time has elapsed and how much execution time is remaining, where the remaining execution time is the difference between the total execution time E_t and the elapsed executed time E_{t-k}. The execution time is directly related to the time taken to execute instruction counts on the processor running at certain clock rate and resource availability.

Case1:

Theorem1:

A set of task represented by T (A_i, E_t, D_i) is readily schedulable in the processor if the utilization of the processor is found zero i.e. U (n^{th}) =0;

A processor is said to be in zero utilization only if it has completed executing its task and no task are in queue waiting to execute or no task are scheduled to run.

In our case after initial scheduling on M processor only chance for any of our M processor to be in zero utilization is due to task completion. If such processor with zero utilization is found then new task with execution time E_t is scheduled to run on it

Theorem 2:

A set of task represented by T (A_i, E_t, D_i) is schedulable in the processor if the utilization of the processor is less than one i.e. U (nth) < 1

The utilization of a processor is found by

$$\frac{E_t - E_{(t-k)}}{} = u\,(n^{th})$$

Where U (n^{th}) is utilization of n^{th} processor in set a set of M processor and E_t-E_{t-k} is the remaining time to finish a task, D_{j-t} is the remaining deadline at the instant new task arrives.

But other than this condition for scheduling tasks in a processor the condition to be satisfied are given in case 2.

Case 2:

When new task arrives if the processors are busy executing their previous task, the remaining execution time E_t-E_{t-k} for every processor is calculated,

Condition 1:

Consider we have the remaining execution time E_t-E_{t-k} of tasks executing in processors to be both lesser and greater than the execution time of new task found by intelligent scheduler considering the resource availability. Then the processor executing task with least remaining execution time E_t-E_{t-k} within multiprocessor and task with highest execution time within the job is chosen.

Condition 1a:

If this least remaining execution time E_t-$E_{(t-k)}$ is less than the execution time E_t of new task then schedule is made such that new task is scheduled to execute after the task on corresponding processor is complete. This guarantees continuous and quick output since the preemptions of task about to complete would result in longer waiting for result of the task.

Condition1b:

If the remaining execution time is same as the execution time of new task then previous task can either preempted or new task can be scheduled to run after the task executing by comparing the deadline of both task, i.e. if the new task is found to have lowest deadline then it is given higher priority so it preempts the executing task, if the new task is found to have higher deadline compared to executing task then it is scheduled to run after completion of the task.

Condition2:

If this remaining execution time E_t-$E_{(t-k)}$ is more than the execution time in all cases, then the processor executing task with least remaining execution time among all processor is preempted such that the processors with high remaining execution time are allowed to execute with their own task thus not overloading the processor. If it is considered that another new task i.e aperiodic task arrives at the very next moment then the new remaining execution time in corresponding processor at that very moment is the sum of the remaining execution time of previous task and the execution time of new task scheduled,

I.e. New remaining time is

$\sum [E_t\text{-}E_{(t-k)}$ (previous) + E_t (new)]

This new remaining execution time is then compared with other processor.

III. ALGORITHM:

1) Initially tasks are scheduled to run on M processor with no conditions Loop;

2) when new task with Et arrives

2.1 If U (nth) = 0 for a processor

- Schedule processor with new task

2.2 If U (nth) = 0 for many processor

- Schedule task randomly until no processor is ideal

2.3 If U (nth) $\neq$ 0 for all processor, Compare processors for E_t-$E_{(t-k)}$

2.3.1 If E_t-$E_{(t-k)}$ is both more and less in different processor, Find processor with least E_t-$E_{(t-k)}$

a) If E_t-$E_{(t-k)}$ < E_t (new task)

Schedule new task to run after completion of previous task

b) If E_t-$E_{(t-k)}$ = E_t (new task)

If deadline of new task < executing task

Preempt executing task & schedule new task

Else

Schedule new task to run after completion

of executing task

2.3.2 If E_t-$E_{(t-k)}$ in different processor is only more than E_t of new task,

Find processor with least E_t-$E_{(t-k)}$ such that

E_t-$E_{(t-k)}$ > E_t (new task)

- Preempt previous task by new task

New Et-$E_{(t-k)}$ is

$\sum [E_t\text{-}E_{(t-k)}$ (previous) + E_t (new)]

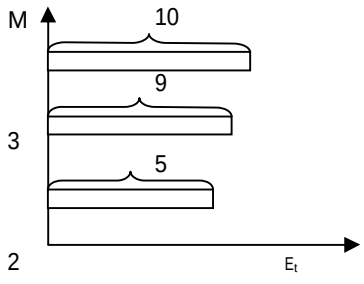
Continue loop;

Example:

Lets consider a multiprocessor system with M set of processor (M=3) and a job with the following tasks arrives.

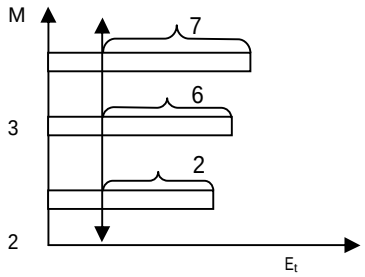
Task	T1	T2	T3	T4	T5	T6	T∞
et	10	9	5	6	3	4	..
Dj	5	8	1	2	6	5	..

Figure a. describes the initial schedule in M processors



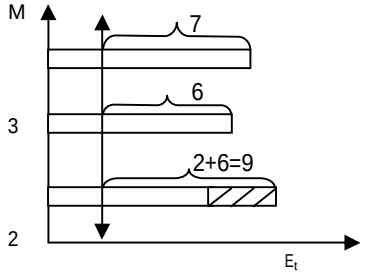
a. Initial schedule of Tm tasks

Figure b. represents the comparison of remaining execution time during arrival of new task tm+i (i=1) with execution time 6 as soon as 3 second elapses.



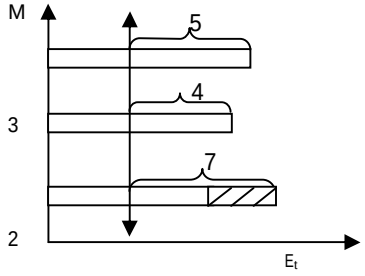
b. Arrival of new task Tm+i (i=1)

Figure c. represents the schedule of new task tm+i satisfying condition 2.3.1.a



c. Schedule of new task -Tm+i (i=1)

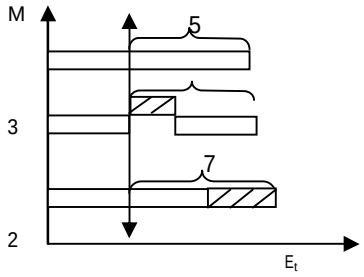
Figure d. represents the comparison of remaining execution time during arrival of new task tm+i (i=2) with execution time of 3 as soon as 2 second elapses.



d. Arrival of new task Tm+i (i=2)

Figure e. represents the schedule of task tm+i (i=2) satisfying condition 2.3.2.
e. Scheduling of task Tm+i (i=2)

Figure f. represents the comparison of remaining execution time during arrival of new task



tm+i (i=3) as soon as 1 second elapses.

f. At the arrival of task Tm+i (i=3)

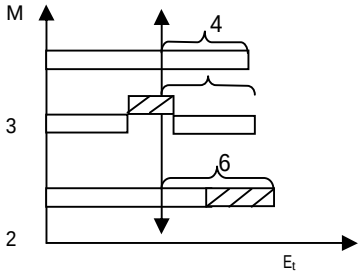
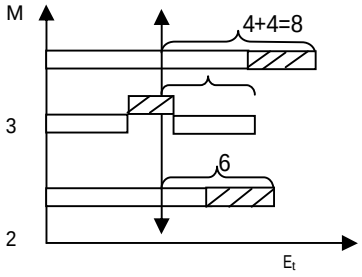


Figure g. represents the schedule of task tm+i (i=3) satisfying condition 2.3.1.b

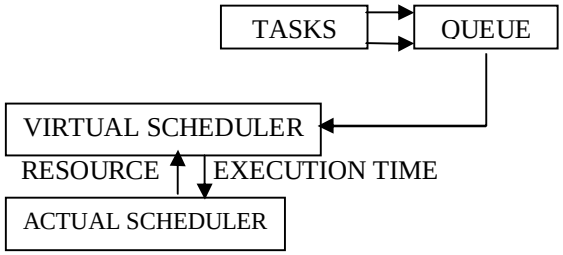


g. Scheduling of task Tm+i (i=3)

IV. IMPLEMENTATION

To schedule tasks based on execution time and resources, the algorithm must know prior to scheduling the execution time a code will take and the amount of resource present to schedule. Execution time of any task can be found only by either running the task or by tracing the entire source code. The number of lines to be traced can vary from tens to several thousands; making it impossible to trace the entire source code. Due to impossibility in tracing the source code, we emphasize the method of running the code in a fictitious environment comparable to the real environment called virtual scheduler. The execution time and remaining execution time are found by using the concepts of virtual scheduler approach and are given to scheduler to schedule the tasks as per our algorithm. The virtual scheduler is run on the same machine running actual scheduler. Initially tasks are made to run on the virtual scheduler in the order of arrival even though two or more task arrives at same time, but the tasks arriving at the same time are notified to actual scheduler to make it schedule tasks considering the remaining execution time to be found by virtual scheduler. While a task is executing if a task arrives, a timestamp in respective timers is made denoting the elapsed time. The remaining execution time is consequently found from the total execution time and the elapsed time. This information's such as remaining execution time and

total execution time are conveyed to actual scheduler to schedule tasks.



Virtual scheduler on cloud network accessing clouds services such as platform as service and software as service is another alternative for implementation where virtual scheduler processes tasks at a remote location accessed by cloud and returns the remaining execution time and the execution time to the scheduling environment.

REFERENCE

[1] C.L.Liu and James W. Layland, "scheduling algorithms for multiprogramming in a hard real time environment" published in journal of ACM (JACM) volume 20 issue 1 - 1973.

[2] Hong, K. and Leung.J, "online scheduling of real time tasks". Appeared in IEEE transaction on computers, volume 41, issue 10, page 1326.

[3] C.M.Krishna and Kang G.shin "real time systems" - international edition 1997.

[4] Damir isovic and Gerhard fohler "Efficient scheduling of sporadic, aperiodic and periodic task with complex constraints" at the proceeding of 21st IEEE real-time system symposium, 2000.

[5] inki hing, miodrag potkonjak and mani B. srivastava "On-line scheduling of hard real time tasks on variable voltage processor" appeared in IEEE/ACM international conference on nov-1998 in page 653.

[6] Bhaskar dasgupta and Michael A. Palis, "online real-time preemptive scheduling of jobs with deadlines on multiple machines" published in APPROX`00 proceedings of third international workshop on Approximation Algorithms for Combinatorial Optimization.

[7] yi-ping you, chingren lee, jenq-kuen lee and wei-kuan shih "Real-time task scheduling for dynamically variable voltage processors" IEEE workshop on power management for real time and embedded systems.

[8] Christopher Clarke and Julie Howrath, "Intelligent Scheduler, Prioritize on Fly".

[9] wei zhao,Krithi ramamritham and john A. Stankovi, "Preemptive scheduling under time and resource constraints", IEEE transaction on computers, Vol c-36, No 8, August 1987, page 949.

[10] Kamaljit Kaur, Amit Chhabra and Gurvinder Singh, "Heuristics Based Genetic Algorithm for Scheduling Static Tasks in Homogeneous Parallel System", international journal of computer science and security 2010, volume 4, issue 2, page 183-198.

[11] Ramamritham. K and Stankovic. J.A., "Scheduling algorithms and operating systems support for real-time systems" proceedings of the IEEE jan 1994, volume 82, issue 1, page 55-67.

[12] Jovanovic.N and bender.M.A, "Task scheduling in distributed systems by work stealing and mugging - a simulation study" 24th international conference on information technology interfaces, 2002, ITI 2002, volume 1, page 259-264.

[13] Oliver Sinnen- "**Task Scheduling for Parallel Systems-** Wiley Series on Parallel and Distributed Computing".

[14] Alejandro Masrur, Sebastian Drossler, Thomas Pfeuffer and Samarjit Chakraborty, "**VM-Based Real-Time Services for Automotive Control Applications"** Proceedings of the 2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications.

[15] Jun Fang, shoubao yang, wenyu zhou and hu song, "Evaluating I/O Scheduler in Virtual Machines for Mapreduce Application" 2010 9th International Conference on grid and cooperative computing (GCC), page 64-69.

[16] Jia tian, yuyang du and hongliang yu, "Characterizing SMP Virtual Machine Scheduling in Virtualization Environment", 2011 international conference on internet of things (iThings/CPScom) and 4th international conference on cyber, physical and social computing, page 402-408.

[17] Forsberg.N, Nolte.T, Kato.S and Asberg.M, "Towards real-time scheduling of virtual machines without kernel modifications", 2011 IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA), page 1- 4.

[18] khalid.o, Maljevic.I., Anthony.R, Petridis.M, Parrott.K, and Schulz.M, "Deadline Aware Virtual Machine Scheduler for Grid and Cloud Computing", 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), page 85-90.

[19] yoginath.S.B and perumalla.K.S, "Efficiently Scheduling Multi-Core Guest Virtual Machines on Multi-Core Hosts in Network Simulation", 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS), page 1-9.

[20] Castrillon.J, Shah.A, Murillo.L.G, Leupers.R and Ascheid.G, "Backend for virtual platforms with hardware scheduler in the MAPS framework", 2011 IEEE Second Latin American Symposium on Circuits and Systems (LASCAS), page 1- 4.

[21] Mohammad I.Daoud and Nawwaf kharma, "A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous processor networks" Journal of Parallel and Distributed Computing, Volume 71, Issue 11, November, 2011.

◈◈◈