Handling Aperiodic Overloads

A seminar paper for Special Emphasis A - Real Time Systems

Yashodhan Vishvesh Deshpande
Hamm-Lippstadt University of Applied Sciences, Lippstadt, Germany
yashodhan-vishvesh.deshpande@stud.hshl.de

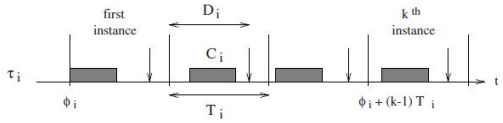

Fig. 1. Periodic Tasks [1]

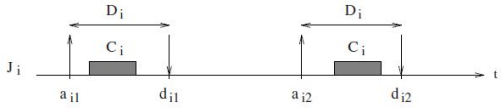

Fig. 2. Aperiodic Tasks [1]

***Abstract*—The ever increasing demand for sophisticated products to improve modern lifestyle has created the need for developing complex real time systems. With the advancement in artificial intelligence the computation requirements of real time systems has grown exponentially. This can result in aperiodic overloads on the computing system which might lead to real time result failures. To avoid these failures it is necessary to have a method that addresses the issue of aperiodic overloads.**

This seminar paper is an in depth dive into methods of handling aperiodic overloads. It discusses the basic concepts related to handling aperiodic overloads and gives an overview of the algorithm needed to handle the overloads. It describes concepts like cumulative value of an algorithm, competitive factor, classification of algorithms, RED algorithm and implementation. It gives an overview of dynamic priority exchange algorithm and slack stealing algorithm.

***Index Terms*—aperiodic, overloads, hard real time systems**

I. Introduction

A. Periodic vs Aperiodic tasks

Periodic tasks are similar to each other in terms of time require to perform the task. Periodic tasks are performed in a fixed schedule since they are identical. Hence the starting time in the scheduling algorithm is at a constant speed. Periodic tasks are shown in figure 1. Aperiodic tasks are also similar to each other in terms of computational times. However they are not performed in a scheduling algorithm. They can be started at any time. Aperiodic tasks are shown in figure 2.

B. Overload and types of overload

In real-time systems load on a particular processing system is calculated by the queuing theory formula. That is

Load = (average processing time of the processes) * (rate of process arrivals)

$$\rho_i(t) = \frac{\sum_{d_k \le d_i} c_k(t)}{(d_i - t)},$$

Fig. 3. Instantaneous load calculation [1]

However this theory does not take into account the time intervals for a particular process. Hence it is only suitable for soft real time systems. For hard real-time systems a more sophisticated theory is needed so that the time intervals are taken into account since failing to meet a deadline can be fatal in hard real time systems.

To measure the load at specific time in a processing system a method was realized by Buttazzo and Stankovic. According to this method, load (p) of every job (J) causing on the system is calculated from the chosen time instance (t) until the deadline of the job (d). The c(t) is the total sum of remaining processing time of the selected job. The algorithm to measure the load is shown in figure 3: The highest load on the processor is the load p(t) of the job that has highest load among all the other jobs in the selected period of time.

A task processing system is overloaded when the total required processing time for the task in a fixed time slot is more than the available time of the processing system in the same time slot. [1]

There are two types of overloads:

1) Transient Overloads: Transient overloads are overloads with p(t) $>$ 1, for a specific period of time of of the whole processing duration. These types of overloads have average load value of smaller than or equal to 1.0 .

This type of overloads is caused in systems that works with event triggered scheduling algorithms. The external interrupts in a event triggered system can cause missing of a deadline due to sudden overload in a short duration. This can cause a serious hazard in a hard real time system as missing a deadline is fatal.

The earliest deadline first is an algorithm in which the tasks are computed in a priority based scheduling. The tasks with the earliest deadline or upcoming deadline is give the highest priority as compared to other tasks with later deadlines.

A processing system designed to work with earliest deadline first scheduling algorithm that is computing aperiodic tasks can run into several problems. One of the problematic phenomenon

is called domino effect. In this phenomenon a hard real time system working reliably and predictably in handling its tasks is interrupted by an aperiodic external event. This event can cause an overload of the system in which the external task can have a higher priority due to its earlier deadline. The external task will be performed earlier than the other regular tasks. This rescheduling of the task priorities can cause missing of the regular task deadlines. Delaying one task can lead to delaying of other tasks in the schedule. This causes a domino effect of missing multiple deadlines of the other tasks in the schedule. [1]

2) Permanent Overload: Permanent overloads are overloads with p(t) $>$ 1 for a uncertain period of time. This type of overload has average value of more than 1.0 for the whole duration of processing. [1]

II. Handling Aperiodic Overloads

A. Cumulative Value of an Algorithm

When an overload of tasks occurs in a real time system then there might be some tasks that might miss there particular deadline. In this situation the aim of the task scheduling algorithm should be to have a degradation of performance in such a manner that more important tasks will be completed on time while the less important tasks will have less priority. This will ensure that the overall system still performing at its level best.

It is necessary to prioritize some tasks because they are critical to the whole system. Tasks like measuring temperature of system pipes in a chemical plant is more important than displaying system graphics on the console.

Every single task is given an arbitrary value base on its importance. The arbitrary value can be decided based on computational time required by the task or some other factors. Whenever the system completes a task within its deadline, the arbitrary value of that completed task can be added to the sum of the values of completed tasks. The sum of the arbitrary values of completed tasks within there deadline is called cumulative value of the scheduling algorithm.

The critical point in this valuation of the scheduling algorithm is that if a deadline of a task in hard real time system is missed then the overall cumulative value is zero. This is because missing a deadline of a task in hard real time system leads to fatal consequences. In soft real time systems the value is still added to the cumulative sum of the values but as time passes after the deadline the value of the incomplete tasks decreases. This is shown in figure 4.

B. Competitive Factor of a Scheduling Algorithm

The competitive factor of an algorithm gives us an estimate of how successfully the algorithm can handle an overload situation. The total maximum cumulative value of all tasks during an overload is called the cumulative value of clairvoyant scheduling algorithm. Clairvoyant scheduling algorithm is a theoretically ideal algorithm which completes all tasks within its deadline.

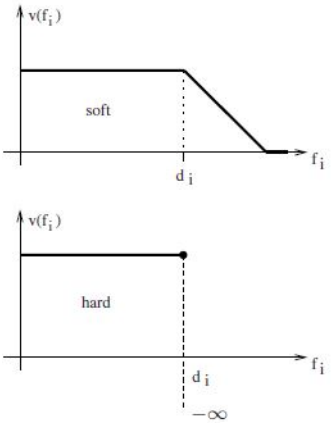

Fig. 4. Cumulative value of tasks with missed deadlines [1]

Fig. 5. Best Effort algorithms [1]

The cumulative value achieved by an algorithm is divided by the cumulative value of the clairvoyant algorithm during the overload situation. This ratio is called competitive factor of a scheduling algorithm. Competitive factor of an algorithm is between 0 and 1. [1]

C. Classification of Algorithms for Overload

Different types of scheduling algorithms used different strategies for handling overload situations. Based on the strategies used the algorithms can be classified into three types.

1) Best Effort algorithms: These types of algorithms don't have any method to know an incoming overload. Every single upcoming task is accepted in the ready queue and is computed once its turn arrives. This is shown in figure 5.

2) Algorithms with Acceptance Test: These types of algorithms perform a test of every single arriving task. The schedulability of every single new task is checked to ensure that the task does not create an overload situation. If the task can be completed within its deadline then it is accepted. If it cannot be completed then the task is rejected. This is shown in figure 6.

3) Robust Algorithms: This algorithm also performs an acceptance test on the upcoming tasks. The tasks that can be completed are moved to the ready queue while the tasks that are not schedulable are moved to the rejected queue. The

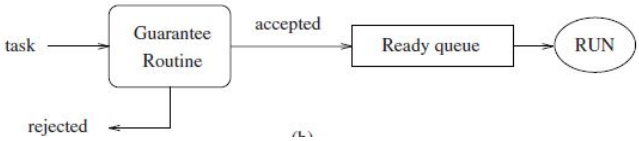

Fig. 6. Algorithms with Acceptance Test [1]

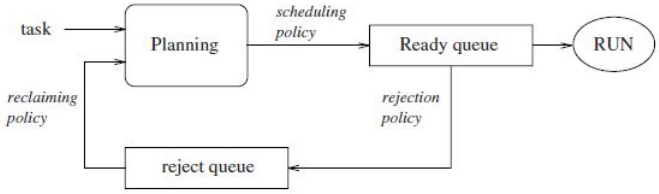

Fig. 7. Robust Algorithms [1]

$$L_i = L_{i-1} + (d_i - d_{i-1}) - c_i(t)$$

Fig. 8. Residual laxity [1]

tasks are rejected or accepted bases of diverse policies that take into account the cumulative value and other factors also. The rejected tasks are not discarded , the are preformed later on when the get scheduled. This is shown in figure 7.

III. Robust Earliest Deadline Algorithm (RED)

The robust earliest deadline algorithm was created to add a rejection test to the original Earliest deadline first algorithm. This was done to increase the cumulative value of the algorithm. The features that it includes are controlled degradation in overloads, deadline tolerance and resource reclaiming. Its is able to point out future deadline misses and the impact it creates on the system.

There are four variables that are used in this algorithm. They are computation time of a task (C) , original deadline of a task (D) , deadline tolerance of a task (M) and importance of a task (v). [1]

The deadline tolerance is amount of time a task is allowed to run after it misses it original deadline. This kind of provides a second deadline for a task. This kind of tasks appear in soft real time system. They are used in multimedia applications. In the RED algorithm the tasks are accepted on the base of there deadline tolerance but are scheduled based on there original deadline. This is done to increase the cumulative value of the algorithm. The schedule of a set of tasks is said to be feasible if all the original deadlines are met while it considered tolerant if all the tasks are completed within there relative deadline tolerance time.

The residual laxity (L) of a task in RED algorithm is interval between its estimated finishing time and its original deadline. This used to carry out the guarantee test to check if a new task is feasible in the schedule. Its calculation is shown in figure 8.

The schedule is considered to be schedulable if the residual laxity is more than or equal to zero for all tasks in the schedule. If the residual laxity is less than zero then the tolerance test is carried out to check if the schedule is tolerant. The tolerance test is only carried out on task that has laxity less than zero. The tolerance test is shown in figure 9.

The schedule is considered to be tolerant if Exceeding time (E)value is equal to zero. Tasks that cause a failure in tolerance

$$E_i = \max(0, -(L_i + M_i))$$

Fig. 9. Tolerance test [1]

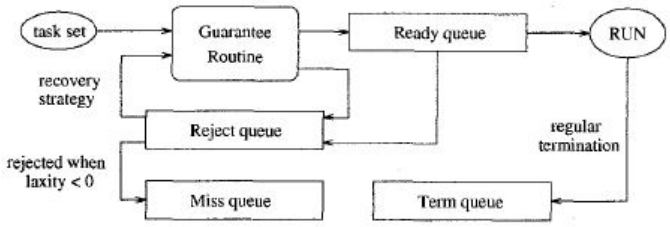

Fig. 10. RED algorithm block diagram [2]

are moved to the missed task queue. [1]This feature of future predictability of tasks allows us to create a better rejection mechanism. The figure 10 shows a block diagram of the RED algorithm.

The C program implementation is carried to simulate the RED algorithm. It calculates the laxity of each task. Task 3 has laxity lower than zero so the task set is not feasible. Then the algorithm calculates tolerance of task 3. The exceeding time is not equal to zero and it is infact smaller than zero. Hence the schedule is not tolerant also and needs to move task 3 to the missed task queue where it will not be computed. It is shown in figure 11.

IV. Overview of other Aperiodic Overload Handling Algorithms

A. Dynamic Priority Exchange Algorithm

The scheduler in this system runs by assigning priorities to tasks. The periodic tasks are run as per EDF. The periodic tasks are of lower priority than aperiodic tasks. Whenever a aperiodic task arrives it is given that higher priority than other periodic tasks. The system is fully utilized to complete the aperiodic task first. [3]

Advantages of this algorithm is that the aperiodic task gets computed completely. Disadvantage of this algorithm is that some periodic tasks might miss there deadlines because of lower priority.

B. Slack-stealing algorithm

In this algorithm the periodic tasks are scheduled by the scheduler. If an incoming aperiodic tasks can be computed

```
PS C:\Users\Yash\Desktop\Emphasis A\Real-time-algorithm> .\test.exe
laxity of job 1 = 1
laxity of job 2 = 0
laxity of job 3 = -1
laxity of job 3  is less than zero, not schedulable
Emax of job 3 = -6
System is intolerent because of task 3 , Emax < 0
PS C:\Users\Yash\Desktop\Emphasis A\Real-time-algorithm>
```

Fig. 11. RED Algorithm implementation

between two periodic task then it is accepted. The aperiodic task is computed and then the remaining periodic tasks are computed. If an incoming aperiodic task cannot be computed between two periodic tasks then it is rejected.

Advantage of this algorithm is that the periodic tasks donot miss there deadline and the aperiodic task overload is handled. Disadvantage of this algorithm is that it cannot handle aperiodic overloads with longer computation times. [4]

V. Conclusion

This paper discusses several key aspects related to handling of aperiodic overloads. It has describes concepts like periodic and aperiodic tasks, overload and its types, cumulative value of an algorithm, competitive factor, classification of algorithm and RED algorithm. The paper also RED algorithm implementation example.It also gives an overview of dynamic priority exchange algorithm and slack stealing algorithm. Also its advantages and disadvantages.

References

[1] Giorgio Buttazzo. *Hard Real-Time Computing Systems*. Springer, 2011. This resource is used to gain primary understanding of all the concepts related to handling of aperiodic overloads. It is used to understand concepts like cumulative value , competitive factor, classification of algorithms and understanding RED algorithm.

[2] Marco Spuri Giorgio Buttazzo Fabrizio Sensini. Robust aperiodic scheduling under dynamic priority systems. This aritcle is used for widening the knowledge of RED algorithm and is specially used for block diagram understanding.

[3] Marco Spuri and Giorgio Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179–210, 1996. This aritcle is used for getting an understanding of dynamic priority exchange algorithm, its advantages and disadvantages.

[4] Thuel and Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *1994 Proceedings Real-Time Systems Symposium*, pages 22–33. IEEE, 1994. This aritcle is used for getting an understanding of Slack-stealing algorithm, its advantages and disadvantages regarding hnadling aperiodic overloads.