Handling Aperiodic Overload

1st Brian Kelein Ngwoh Visas
Electronics Engineering
Hamm-Lippstadt University of Applied Sciences
Lippstadt, Germany
brian-kelein-ngwoh.visas@stud.hshl.de

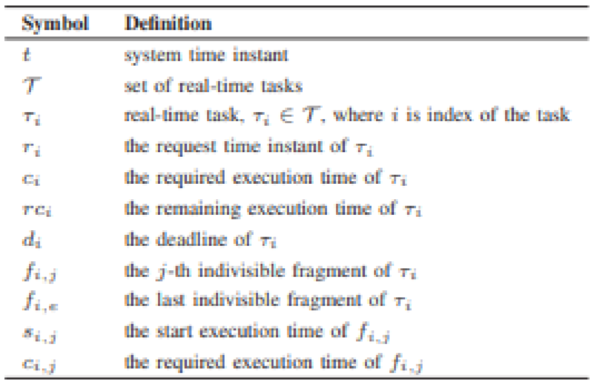
***Abstract*—Text**

***Index Terms*—component, formatting, style, styling, insert**

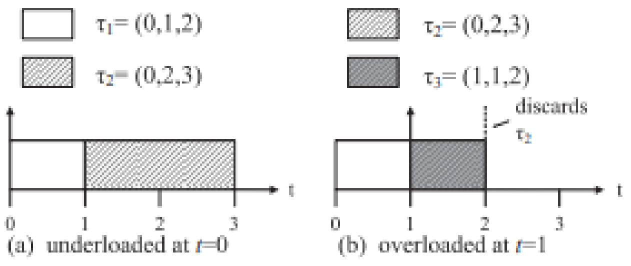
I. Introduction

A. Definitions

The way Systems are designed, most if not all require a modus operandi. In this forms of Operation they all have tasks they have to complete within a particular time to ensure, they function properly, this is also known as their Deadlines. So the main reason why some systems are called Real Time systems is because they are able to meet all the Deadlines, without missing out on any. According to [1] therefore, in contrast to conventional computer systems where the goal usually is to minimize task response times, the emphasis here is on satisfying the timing constraints of tasks.



Symbol	Definition
t	system time instant
$\mathcal{T}$	set of real-time tasks
τ_i	real-time task, $\tau_i \in \mathcal{T}$, where i is index of the task
r_i	the request time instant of τ_i
c_i	the required execution time of τ_i
rc_i	the remaining execution time of τ_i
d_i	the deadline of τ_i
$f_{i,j}$	the j-th indivisible fragment of τ_i
$f_{i,e}$	the last indivisible fragment of τ_i
$s_{i,j}$	the start execution time of $f_{i,j}$
$c_{i,j}$	the required execution time of $f_{i,j}$



[2]

Fig. 1. Example for overloaded and underloaded

Identify applicable funding agency here. If none, delete this.

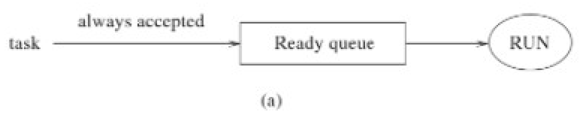
1) An Overloaded System: Generally a system always has tasks to fulfill and this tasks has to be done within a period of time.When a system is not able to meet up with all the deadlines, this reduces its computing power and we refer to such a system as an Overloaded System.Also in due to Overload some tasks are left unattended. But when such tasks are taken up and completed after their deadlines, then the system is said to be an Under loaded system.From [3] there exists a schedule that will meet every task.

2) Aperiodic Tasks: Aperiodic in this context refers to tasks that have no idea of future instances, when scheduling. These tasks are assumed to be independent of each other and of the Offline scheduled tasks.They have the following characteristics, Time, Firm absolute deadline and Value. Most of this is unknown at design time, and aperiodic tasks can also be used to handle non-critical periodic activities. [4]

3) Value: This is a measure of the benefit to the system associated with completing the task in time. Only aperiodic tasks are associated with values, since offline scheduled tasks are never considered for rejection when resolving overload situations. The values are considered to be cumulative, i.e., two sets of tasks can be compared by their respective sum of values. Tasks contribute with their value to the system if they finish in time, otherwise they do not contribute at all. In this paper we assume static values ranging from 1 to Max Value, where a higher value indicates a greater benefit. [4]

4) Typical Scheduling Schemes: Based on the problems that arise due to overloads, Academic enthusiasts have come up with Algorithms whose main aim is to predict and handle these overloads and in [5] it is divided into three main classes.

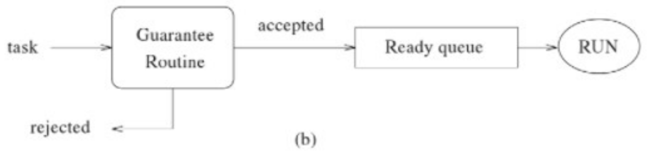
a) Best Effort: In this category Algorithms have no prediction for overload. when a new task arrives, it is taken directly into the ready queue. Here system performance would be properly controlled through priority assignment,which takes task values into consideration.



[5]

Fig. 2. Best Effort

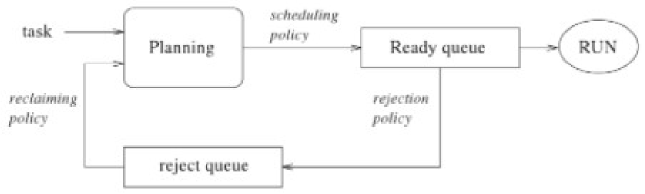
b) With Acceptance: Here we deal with Algorithms that have an admission control, and a guarantee test is performed at every job activation. Every time there is an incoming task the schedulability of those set of tasks is verified based on worst-case assumptions, and this routine is almost always guaranteed. When a set of task are schedulable then it is accepted into the system. else it is rejected.



[5]

Fig. 3. With Acceptance

c) Robust: This category includes those algorithms that separates sets of tasks by timing constraints and importance. In doing this two policies are considered one for task rejection and the other for task acceptance. With this category when a new task enters the system, the schedulability based worst-case assumption is verified. Once found schedulable it is accepted into a ready queue, else rejected.



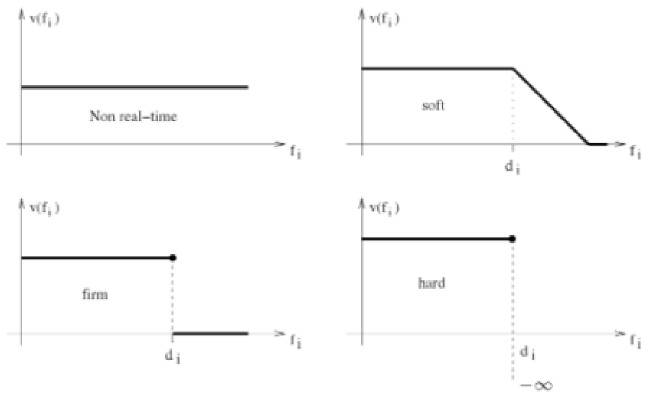
[5]

Fig. 4. Robust

B. Some Important Terms

1) Performance metrics: when tasks are activated dynamically an overload occurs.there are no algorithms that can guarantee a feasible schedule of set task Since one more tasks will miss their deadlines, it is preferable that late tasks be the less important ones in order to achieve graceful degradation.Distinguishing between time constraints and importance for a set of tasks is really very crucial for an overloaded system.For a system to emphasize importance, each task is associated to an additional parameter which its Value. This value is important with respect to value such that most times this value is set to computation time. And other times set to an arbitrary integer number or the ratio of an arbitrary number and the task computation time which can be referred to as Value density as analysed by Buttazzo in [5]. This value can be used by the system to make scheduling decisions.

The graphs above describe Utility functions that can be associated to a task to describe its importance.In [5] When the importance of a task set has been defined, then we can measure the performance of a scheduling algorithm by accumulating the values of a task utility functions computed



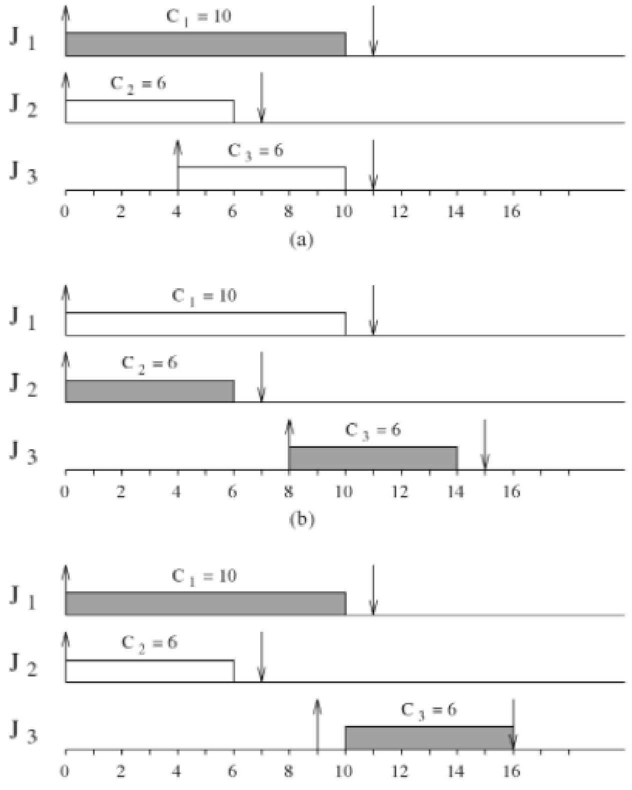
[5]

Fig. 5. Utility functions that can be associated to a task to describe its importance

at their completion time.B is defined as the cumulative value of a scheduling algorithm

$$\Gamma_B = \sum_{i-1}^{n} v(f_i) \tag{1}$$

Below are some analysis of maximum values in [5]



[5]

Fig. 6. No optimal online Algorithm exist in overload conditions, since the schedule that maximizes Γ depends on the knowledge of future arrivals $\Gamma_{max} = 10$ in case (a), $\Gamma_{max} = 12$ in case (b) and $\Gamma_{max} = 16$ case (c).

Furthermore a scheduling algorithm is said to be at its Optimum, when it is able to maximize the value achievable on a task set.

2) Effective Value Density:

3) Competitive factor: The cumulative value obtained by a scheduling algorithm on a task set is a representation, of the measure of the performance for that particular task set. A parameter that measures the worst case performance of a scheduling algorithm is also known as the competitive factor, introduced by Baruah et al. [3]

a) Definition:

II. Handling Overloads

A. Overload Handling

1) Problem Formulation:

B. Scheduling schemes with example Algorithms

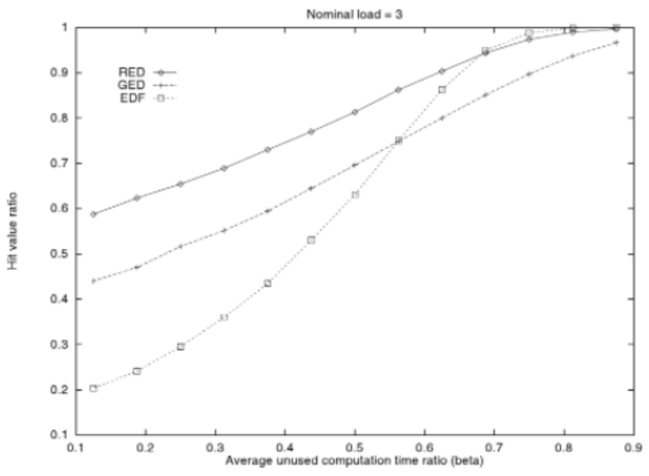
1) ROBUST Algorithm (Best Effort):

*2) D*over Algorithm(With Acceptance):*

3) RED Algorithm (Robust scheme):

4) SMT Based Scheduling:

C. Performance Evaluation of Algorithms(Comparison)



[5]

Fig. 7. Performance of various EDF scheduling schemes: Best Effort(EDF), guaranteed (GED), and Robuat (RED)

III. Application of Handling Overloads in FTT-CAN Protocol

A. FTT-Can Protocol

B. Problem Formulation

C. Overload management

D. Schedulability analysis

In [6] we can see here the author performs schedulability analysis of both periodic and aperiodic messages by exploiting time analysis on a cycle by cycle basis. This results in an iterative schedulability test, which can hardly be used at runtime.

IV. Conclusion

References

[1] S. Baruah and J. Haritsa, "Scheduling for overload in real-time systems," *IEEE Transactions on Computers*, vol. 46, no. 9, pp. 1034–1039, 1997.

[2] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "Scheduling overload for real-time systems using smt solver," in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2016, pp. 189–194.

[3] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload," in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, 1991, pp. 100–110.

[4] J. Carlson, T. Lennvall, and G. Fohler, "Enhancing time triggered scheduling with value based overload handling and task migration," in *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003.*, 2003, pp. 121–128.

[5] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[6] F. Bertozzi, M. Di Natale, and L. Almeida, "Admission control and overload handling in ftt-can," 10 2004, pp. 175 – 184.