PRES LUNAM
École Doctorale STIM
Sciences et Technologies de
l'Information et Mathématiques

**Spécialité :** Informatique
**Laboratoire :** IRCCyN
**Équipe:** STR

# BGW model: An approach for overload management in fault tolerant real-time systems

OULD SASS, MOHAMED
Mohamed.Ould-Sass@irccyn.ec-nantes.fr

**Abstract:** In this paper, we propose a model for the management of overload in fault tolerant real-time systems. This model is based on the idea that each task has two independent versions, a primary version and a secondary version. We assume that the system is overloaded and we must overcome this situation by eliminating some task instances but in a well determined and specified way. We must respect a set of conditions imposed to guarantee a minimum system operation, during transient overload while offering a non-degraded execution under normal conditions.

**Keywords:** *Real-time system, overload management, fault tolerance, uniprocessor system*.

## 1 Introduction

A real-time system is a reactive system with an external environment or a process through a set of sensors and actuators. Sensors deliver measures about the environment or the process while communicating actuators receive commands and execute them. Real-time systems are classified according to the importance of the time constraints and the consequences that can arise when they are not satisfied. Overload management and fault tolerance are two fundamental properties in a real-time system. In this work we are interested to study and design new task models and these capabilities. A scheduling algorithm in a real-time system is responsible for the management of processor occupancy in the time under different constraints: time, fault tolerance and load management. Fault tolerance is to keep the system operation in the presence of faults even if the results produced are of. Faults in a real-time system are divided into permanent faults, transient and intermittent faults. A permanent fault is a fault that lasts and requires a repair operation. It can be either hardware or software. A transient fault is a fault operation that appears temporarily, due to an external disturbance or a transient overload of the processor. Intermittent fault is a transient fault that occurs in a sporadic or regular manner. The fault tolerance techniques are based on the principle of redundancy. There are several types of redundancy: hardware redundancy, software redundancy and temporal redundancy. Generally to cover permanent faults, the best is to use hardware redundancy. For transient and intermittent faults, the temporal redundancy techniques are applied. The temporal redundancy consists in multiplying of task execution over time. That is to say, when encountering a fault, we re-execute the same task. If we repeat the execution of the same version of the task, we are in a situation of pure temporal redundancy. If we repeat the execution of the task, but with a different version, (a shorter one for example), we can say that it is a hybrid software and time redundancy. Our works target the resolution of transient overloads through time redundancy.

## 2 Background and related works

Our approach is essentially based on the combination of two task models: the deadline mechanism model [2][3][10] and the skip-over model [8][14].

### *2.1 Scheduling for fault-tolerance*

The basic idea of the deadline mechanism model [10] is to consider that every task have two independent versions: a major version (primary) and a version with degraded quality (secondary version) [2][3][10]. The primary version contains more functions and produces results with a higher quality of service when it is properly and completely executed. However, it can have an unknown computation time which can sometimes be infinite [2][3][10].The reliability of this version is not always guaranteed, because the complexity of its functions and its high level of resource utilization.

The secondary version contains the minimum required and the minimal functions to be executed. It produces results with less precision, but still acceptable. This version has a shorter execution time than the primary one. It is well-known and reliable, given the simplicity of the functions to be executed [2][3][10].

### *2.2 Scheduling for overload management*

Among the basic methods for overload management we can cite the skip-over model [8] and the (m, k) firm model [6] [16]. The idea of skip-over model is to characterize the tolerance of each task to occasionally skip the execution of some instances. The author use a factor ($s_i$), $2 \leq s_i \leq \infty$ which means that on ($s_i$) successive instances, we can have only one skipped [8] [14].
A task is divided into two types of instances where each type occurs during a single period of the task. Every instance of a task can be either red or blue. A red task instance must complete before its deadline while a blue task instance can be aborted at any time [8]. A blue task instance may complete by its deadline or miss its deadline. When a task misses its deadline we say that the task (or deadline) instance was skipped [8].
The distance between two consecutive skips must, be at least $s_i$ periods. That is, after missing a deadline at least ($s_i - 1$) task instances must meet their deadlines [8].
With respect to the (m, k) firm model, it consists in tolerating (m) instances skips over an interval of (k) successive instances [6][9][15][16].

# 3 BGW model

### *3.1 Description of the model*

BGW model (Black, Gray, and White) model is a combination of two techniques: the deadline mechanism model and the skip-over model. In our model, we assume that we have a set of periodic tasks with the following assumption:
- Each instance of a task can be either Black, Grey or White.
- Each instance of a task must have two versions, a primary and secondary one.
- A Black instance means that we must execute its primary version (critical instance).
- A Gray instance means that we must execute either its secondary or primary version (fault-tolerant instance).
- A White instance means that we do not have necessarily to execute any version (the instance can be skipped).
- The execution time of the secondary version is less than that of the primary version.

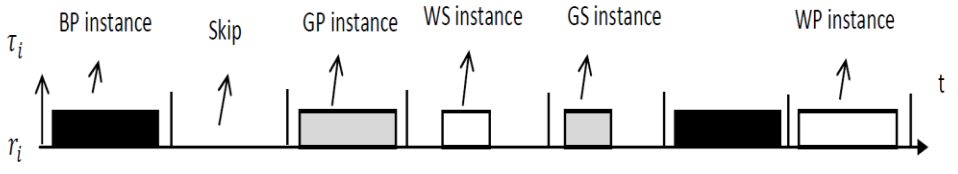


Fig.1) Illustration of different instances in the BGW model.

The Quality of Service of BGW tasks can be measured using two performance metrics:
- A qualitative metric (i.e. the number of primary version executed over the total number of instances successful executed).
- Quantitative metric (i.e. the number of instances successful executed over the total number of instances).

We define the qualitative performance by the number of secondary versions executed and the quantitative performance by the number of instances losses for a task.
We have a model with two constraints types: qualitative degradation and quantitative degradation constraints (loss). To model these constraints, we must add at least two independent parameters to the well-known periodic tasks model of Liu and Layland [11].
We propose two modeling possibilities.

### *3.2 BGW1 model*

So, we represent each task as:

$$\tau_i\left(r_i, T_i, D_i, C_i^p, C_i^s, n_i, l_i\right), \text{ where: (2)}$$

$r_i$: Is the release time of the task.
$T_i$: His period.
$D_i$: His deadline.
$C_i^p$: The execution time of his primary version.
$C_i^s$: The execution time of his secondary version.
$n_i$: The period of execution of the obligatory primary version of a task.
$l_i$: The period of execution of the obligatory secondary version of a task, at the interior of the interval between two successive executions of obligatory primary versions.

### *3.2.1 Feasibility test in BGW1*

Theorem 01 (necessary condition):
A set of periodic and independent tasks $\Gamma = \{\tau_i(r_i, T_i, D_i, C_i^p, C_i^s, n_i, l_i),\ 1 \leq i \leq n\}$, modeled by BGW1, is schedulable only if:

$$\sum_{i=1}^{n} \frac{1}{n_i T_i} C_i^p + \frac{1}{n_i T_i} \left\lfloor \frac{n_i - 1}{l_i} \right\rfloor C_i^s \leq 1 \quad (3)$$

Proof. Submitted to publication.

### *3.3 BGW2 model*

So, we represent each task as:

$$\tau_i(r_i, T_i, D_i, C_i^p, C_i^s, n_i, s_i), \text{ where:} \quad (4)$$

$r_i$: Is the release time of the task.
$T_i$: His period.
$D_i$: His deadline.
$C_i^p$: The execution time of his primary version.
$C_i^s$: The execution time of his secondary version.
$n_i$: The period of execution of the obligatory primary version of a task.
$s_i$: It is as the loss factor of Skip-over model, but between the secondary versions in here, that is to say, the minimum distance between two consecutive losses in secondary versions.

### *3.3.1 Feasibility test in BGW2*

Theorem 02 (necessary condition):
A set of periodic and independent tasks $\Gamma = \{\tau_i(r_i, T_i, D_i, C_i^p, C_i^s, n_i, s_i),\ 1 \leq i \leq n\}$, modeled by BGW2, is schedulable only if:

$$\sum_{i=1}^{n} \frac{1}{n_i T_i} C_i^p + \frac{s_i - 1}{s_i} C_i^s - \frac{1}{T_i} \left(\frac{1}{n_i} - \frac{1}{LLCM(n_i, s_i)}\right) C_i^s \leq 1 \quad (5)$$

Proof. Submitted to publication.

## 4 Conclusion

In this work, we proposed a new task models for overload management in fault tolerant real-time systems. We studied the problem of checking the feasibility for such systems. Future work includes designing some on-line scheduling algorithms of these models.

## References

[1] G.-C. Buttazzo, Hard Real-Time Computing Systems, Kluwer academic, 1997.
[2] Housein Chetto and Maryline Chetto, An adaptive scheduling algorithm for fault-tolerant real-time system, Software engineering journal May 1991.

[3] Houssein Chetto and Maryline Chetto, Some results of the earliest deadline scheduling algorithm, IEEE Transactions on Software Engineering, 15.(10) October 1989.
[4] Ching-Chih Han, Kang G. Chih,Wu Jian, fault-tolerant scheduling algorithm for real-time periodic tasks with software faults, IEEE Transactions on computers, 52. (3) March 2003.
[5] T. Garcia-Fernandez, Conception et développement de composants pour logiciels temps-réel embarqués, Thèse de Doctorat, Université de Nantes, 2005.
[6] Hamdaoui, M., Ramanathan, P., A dynamic priority assignment technique for streams with (m, k)-firm deadlines, IEEE Transactions on Computers, 44(4): 1443-1451, 1995.
[7] K. Jeffay and D.L. Stone, Accounting for interrupt handling costs in dynamic priority task systems, In Proceedings of the 14th IEEE Real-Time Systems Symposium, pages 212.221, December 1993.
[8] G. Koren and D. Shasha, Skip-over: Algorithms and complexity for overloaded systems that allow skips, IEEE Real Time Systems Symposium, December 1995.
[9] A. Koubaa, Gestion de la Qualité de Service temporelle selon la contrainte (m, k)-firm dans les réseaux à commutation de paquets, Thèse de doctorat, Institut National Polytechnique de Lorraine, 2004.
[10] A.L. Liestman and R.H. Campbell, A fault-tolerant scheduling problem, IEEE Transactions on Software Engineering, 12(11):1089-95, November 1986.
[11] C. Liu and J. Layland, Scheduling algorithms for multiprogramming in real- time environment, Journal of ACM, 1(20) :46-61, October 1973.
[12] J.-P. Lehozcky, L. Sha, Y. Ding, The rate-monotonic scheduling algorithm: exact characterization and average case behaviour, In Proceedings of the IEEE Real-Time Systems Symposium, pp. 166-171, 1989.
[13] J.-T. Leung and J. Whitehead, On the complexity of fixed-priority scheduling of periodic real-time tasks. Performance Evaluation, 2, pages 237- 250, 1982.
[14] A. Marchand, Real-time scheduling with quality of service constraints. Theory and integration, PhD thesis, University of Nantes, October 2, 2006.
[15] G. Quan, X. Hu, Enhanced fixed-priority scheduling with (m, k)-firm guarantee, In Proceedings of the 21st IEEE Real-Time Systems Symposium (RTSS'00), Orlando, Florida (USA), pp. 7988, 2000.
[16] P. Ramanathan, Overload management in real-time control applications using (m, k)-firm guarantee, IEEE Transactions on Parallel and Distributed Systems, 10(6), 1999.
[17] M. Silly-Chetto, Sur la problématique de l'ordonnancement dans les systèmes informatiques temps-réel, Rapport d'HDR, Université de Nantes, 1993.
[18] Y.-Q. Song, A. Koubaa, Gestion dynamique de la QdS temps-réel selon (m, k)-firm, École Temps Réel (ETR'03), Toulouse, 2003.
[19] W.-K. Shih, W.-S. Liu, Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error, In Proceedings of the IEEE Transactions on Computers, 44(3), 1995.
1982