BGW: A Novel QoS Model for Firm Real-time Computing Systems

Mohamed Ould Sass, Maryline Chetto, Audrey Queudet

BGW: A Novel QoS Model for Firm Real-time Computing Systems

Mohamed Ould-Sass, Maryline Chetto and Audrey Queudet
University of Nantes - IRCCyN Institute
Nantes, France
e-mail: maryline.chetto@univ-nantes.fr

***Abstract*—Tasks in a real-time computing systems are commonly periodic. Each job generated by the invocation of a periodic task has normally a deadline constraint by which it must complete its execution in all circumstances. However, the hardware may be subject to failures, faults may be present in software, energy to supply the system may be depleted and the processor may be overloaded transiently. Any of these exceptions involves a situation where it is no longer possible to meet the deadlines of all the jobs in the application.**
In this paper, we discuss a general model for the application task set that permits the operating system to correctly manage the above emergency situations in accordance with specific parameters statically attached to every task in addition to its classical timing parameters. The model is inspired by the well-known Deadline Mechanism and Skip-Over model. We introduce the BGW model where each job of any periodic task can get one of the three colours Black, Grey and White. A colour specifies that the job has to imperatively execute the primary version, the job has to execute at least one version among primary and alternate or the job may be discarded. We briefly discuss implementation issues for this new model.

I. Introduction

For real-time systems, worst-case timing models are employed to validate whether timeliness properties generally expressed in deadlines meeting, can be satisfied. In the *hard real-time* computing context, it is imperative that all the jobs generated by individual invocations of a periodic task meet their deadline. Nonetheless, many applications can tolerate bounded non-satisfaction of their timeliness properties due to inherent robustness of the system. Applications such as video processing are characterized by highly fluctuating and content-dependent execution times and there is a tolerance for deadline misses limited to a specified upper bound. Identically, for control engineering applications, some deadlines can be missed without compromising correct functioning because the physical process to be controlled as well as the control algorithms implemented in the tasks have inherent robustness and make it possible to miss deadlines up to a known bound.

In that paper, we consider the *firm real-time* systems (also called weakly-hard) where timeliness properties normally guaranteed may admit irregularities in occasional situations. We focus on the following causes: hardware failure, software bugs, energy depletion and prohibitive execution times. In the remainder of the paper, *an exception* will refer to any such abnormal situation that can provoke deadline missing. We need a framework that allow real-time systems to graciously adapt to exceptions by adjusting their Quality of Service (QoS).

A firm real-time system is qualified as *correct* even if it does not meet its timing properties. However, it is necessary to perform an analysis in order to firstly estimate the frequency and impact of the exceptions that may lead to deadline missing and secondly to make that metrics be available to assess the overall QoS of such a system. In general, the QoS is measured in terms of computation quality and deadline miss ratio. In our work, they are referred to qualitative metrics of QoS and quantitative metrics of QoS.

When the timing constraints cannot be met with computation-quality, one way of maintaining an acceptable QoS is to trade computation-quality for timeliness. We then propose the Deadline Mechanism [7]. In the Deadline Mechanism, two versions of programs are provided for each task: primary and alternate. The primary version is the normal program that produces good quality results. But its execution is prone to failures (possible software bugs, unbounded wait for sensor data, etc). In contrast, the alternate version contains only the minimum required functions and produces less precise but acceptable results with no possible failure. In this paper, the Deadline Mechanism is integrated in our model for providing fault-tolerance.
A common method for dealing with overload conditions is to reduce the load by skipping some jobs. Overload commonly is due to prohibitive execution times. But overload can appear in case of supply shortage which appears when the available energy is not sufficient to execute all the periodic load. The effectiveness of the approach called *Skip-Over* has been demonstrated especially in multimedia applications [6]. We propose to integrate the Skip Over model in our QoS model in order to cope with overload conditions.

Contribution. This paper introduces a systematic approach for modeling QoS requirements of firm real-time systems. We will propose a novel QoS model wherein timing requirements can be violated during exceptions up to an acceptable known bound. In our model, this bound will be characterised in terms of two metrics.

Organization. The remainder of the paper is organized as follows. Background materials on periodic scheduling with overload and fault-tolerance considerations are presented in

Section II. Section III describes our novel QoS model called BGW. In section IV and V successively, we describe the two variants BGW1 and BGW2. We briefly discuss the framework of the scheduler needed by the BGW model in section VI. Section VII concludes the paper.

II. Background Materials

A. The classical periodic task model

The vast majority of work relative to real-time scheduling considers the following basic model. A periodic task τ_i is characterized by four-tuple (r_i, C_i, D_i, T_i). It respectively gives the release time of the first job of τ_i called *offset*, worst case execution time (normalized to processor computing capacity), relative deadline and period of jobs of τ_i. τ_i generates an infinite succession of jobs that arrive at each instant $(r_i + kT_i)$ for all integer $k \geq 0$. Every job of τ_i must complete within D_i time units from the time it was issued. It is assumed that $0 < C_i \leq D_i$. In what follows, a request is the invocation of a periodic task. The litterature mainly considers constrained-deadline task sets in which every τ_i has its relative deadline no larger than its period (i.e. $D_i \leq T_i$), implicit-deadline task sets where $D_i = T_i$) for every τ_i and synchronous task sets where all the first jobs of all tasks release simultaneously. This classical model is the basis for our QoS model.

B. Real-time Scheduling issues

Real-time scheduling theory has been primarily concerned from several decades with obtaining solutions to firstly, the feasibility analysis problem and secondly the run-time scheduling problem [8]. In the first problem, we aim to determine whether there exists a schedule where the periodic tasks will meet all deadlines. In the second one, we want to determine a scheduling algorithm that successfully schedules a task set which is deemed to be feasible. Most of these works assume that there are no faults present in the system and no possibility of processing overload.
Within the context of preemptive uniprocessor scheduling of periodic task sets, it has been shown that the Earliest Deadline First scheduling algorithm (EDF) is an optimal scheduling algorithm [8]. At each time instant, it chooses for execution the ready job with the smallest deadline (with ties broken arbitrarily). EDF is consequently the algorithm of choice under normal functioning since any feasible task set is guaranteed to be successfully scheduled by EDF. However, the feasibility analysis problem turns out to be less straightforward because any computing system could be subject to unpredictable situations that can stop the scheduler from guaranteeing all the deadlines. In order to make this scheduling algorithm resilient with exceptions which are failures and overload mainly, the algorithm must be combined with specific techniques first to recover from failures and second to cope with transient overload.

C. Overload Management

Several approaches have been proposed to address deadline missing in firm real-time systems. In the (m,k)-firm model, at least m jobs out of any k consecutive jobs from the same task must meet their deadlines for correct functioning [4]. The elastic task model is an attractive model for adapting real-time systems in the presence of overload [1]. The method is to reduce the load by enlarging activation periods. Tasks'periods are considered as springs and can change to adapt the QoS so as to keep the system underloaded. The Skip-Over model can also be used to handle overload conditions [6]. Koren and Shasha look at the problem of uniprocessor overload by authorizing occasional deadline violations in a controlled way. A periodic task τ_i is characterized besides its basic paramaters by a skip parameter s_i. This parameter represents the tolerance of this task to miss deadlines. The distance between two consecutive skips must be at least s_i periods. When s_i equals to infinity, no skips are allowed and τ_i is a hard periodic task. Every job of a task is either *red* or *blue*. A red job must complete before deadline whereas a blue job can be aborted at any time.
The following two scheduling algorithms were presented in [6].

- the Red Tasks Only (RTO) algorithm where Red jobs are scheduled as soon as possible according to Earliest Deadline First algorithm while blue ones are always rejected.
- the Blue When Possible (BWP) algorithm which is an improvement of the previous one. BWP schedules blue jobs whenever their execution does not prevent the red ones from completing before deadlines. In other words, blue jobs are served in background relatively to red jobs.

In [9], Queudet-Marchand and Chetto continued this work with proposing the RLP (Red as Late as Possible) and RLP-T scheduling algorithms that outperform the prior ones. We propose to integrate the Skip Over model in our QoS model so as to cope with possible overload conditions.

D. Fault-tolerance Management

Some works have developed specific algorithms or approaches to combine fault-tolerance and scheduling from the beginning of the eighties. In [7], Liestman and Campbell propose a Deadline Mechanism that can guarantee that a primary task will meet its deadline if there is no failure, and that an alternative task of less precision will run by the deadline if there is a failure. If the primary task executes then it is not necessary to run the alternative task called *alternate* or *back-up*. And the time set aside for the alternate is reused. The objective of the scheduling algorithm is to guarantee either the primary or the alternate version of each task to be correctly completed before the corresponding deadline while trying to complete as many primaries as possible. Liestman and Campbell studied the aforementioned fault-tolerant scheduling problem under the assumption that the task set is simply periodic, i.e., the period of each task is a multiple of the next smaller period.
Chetto and Chetto continued with this work in [3]. Specifically, they show how to quickly switch to a new task schedule upon failure, where that new schedule has been precomputed offline.

In this approach they ensure that hard deadlines are met by alternates in the face of failures while the chance of success for the primaries is maximized. The embedded alternate schedule is not used unless there is a failure.
The so-called *Last Chance strategy* was proved to outperform the First Chance strategy that first guarantees the feasible execution of the alternates. Any alternate starts its execution at the latest possible time. At runtime, the primaries are scheduled during the remaining intervals before their alternates. The alternates can preempt a primary when a time interval reserved for the alternates is reached. Whenever a primary is completed successfully, the execution of its corresponding alternate is no longer needed and, hence, an online scheduling algorithm must dynamically deallocate the time interval(s) reserved for the alternate so as to increase the processor time available for the execution of other primaries.
The algorithm is based on an offline EDF version, called Earliest-Deadline-first as Late as possible (EDL) [3], to reserve time intervals for the alternates. Then, at runtime, any online preemptive scheduling algorithm schedules the primaries. The reconstruction is achieved by removing the alternate corresponding to the completed primary. Then the remaining alternates are rescheduled from the start time of the EDL schedule up to a specific point bounded in time by the end of the current hyperperiod. It was demonstrated in [2] how to reduce the online overhead of the reconstruction procedure.

In [5], Han and Shin also follow the last chance strategy except that it is based on fixed priority-driven preemptive scheduling, such as the Rate-Monotonic (RM) algorithm.

This approach for fault-tolerance based on dynamic passive redundancy reveals suitable for many applications which must provide for predictability while reacting quickly to occuring faults. We propose to integrate the Deadline Mechanism in our QoS model.

III. Description of the general BGW model

In this section, we present a novel QoS model called Black-Grey-White model, BGW for short. We deal with the problem of scheduling sets of firm periodic tasks capable of handling exceptions such as faults (software or hardware) and overload (due to time or energy starvation).

The BGW model integrates two approaches:

- the Deadline Mechanism where each real-time periodic task uses two independent versions for the purpose of meeting timing constraints while tolerating faults.
- the skip-over model where each real-time periodic task has a skip parameter.

Every job generated by a periodic task may have one of the three following colours :

- *Black* if the job has to imperatively produce a result with the primary version
- *Grey* if the job has to imperatively produce a result with at least one version, in preference the primary
- *White* if the job may be dropped i.e. the job has no execution requirement even if it is preferable to execute one of the two versions

Formally, the model is characterized by a periodic task set τ comprised of n tasks: $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ as described in section II-A. τ is completely-specified since all parameters of each job can be determined prior to system run-time.

Each task τ_i is characterized by :

- its five *timing parameters* $(r_i, C_i^a, C_i^p, D_i, T_i)$ where C_i^a respectively C_i^p denotes the execution time of the alternate respectively the primary
- its two *QoS parameters* according to the underlying variant as described below

There are several versions for the BGW model. In this paper we will describe only two of them.

IV. The BGW1 variant

In the BGW1 model, every task τ_i is characterized by the following two QoS parameters:

- n_i : Upper bound on the number of requests between two consecutive successful executions of the primary version.
- l_i : Upper bound on the number of requests between two consecutive successful executions of a job (either the primary or the alternate)

Consequently, any BGW1 task τ_i is characterized by $(r_i, C_i^a, C_i^p, D_i, T_i, n_i, l_i)$.
We may derive the following feasibility test i.e. the necessary condition for the task set to be schedulable in the BGW1 model. The worst case situation is when primaries regularly execute with a constant period equal to $n_i.T_i$ and when any primary or alternate execution is distant from any primary or alternate execution with a constant period equal to $l_i.T_i$.

Theorem 1: (BGW1 Feasibility test) A BGW1 task set is feasible on monoprocessor only if

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

Proof. Submitted to publication.

V. The BGW2 variant

In the BGW2 model, every task τ_i is characterized by the following two QoS parameters:

- n_i : Upper bound on the number of requests between two consecutive successful executions of the primary version.
- s_i : Lower bound on the number of requests between two consecutive skips (called skip factor) of the alternate version

Consequently, any BGW2 task τ_i is characterized by $(r_i, C_i^a, C_i^p, D_i, T_i, n_i, s_i)$.
We may derive the following feasibility test i.e. the necessary condition for the task set to be schedulable in the BGW2

model. The worst case situation is when primaries regularly execute with a constant period equal to $n_i.T_i$ and when the skip of a job is distant from the next skip with a constant period equal to $(s_i - 1).T_i$.

Theorem 2: (BGW2 Feasibility test) A BGW2 task set is feasible on monoprocessor only if

$$\sum_{i=1}^{n} \frac{1}{T_i}\left(\frac{1}{n_i}C_i^p + \left(\frac{s_i - 1}{s_i} - \frac{1}{n_i} + \frac{1}{LCM(n_i, s_i)}\right)C_i^s\right) \leq 1 \quad (2)$$

Proof. Submitted to publication.

VI. Scheduling algorithms

Any scheduler for the BGW model must:

- Guarantee the constraints of
  - the BGW1 model defined by the execution of at least one primary version over n_i successive requests, and one alternate version over l_i successive requests
  - or the BGW2 model defined by the execution of at least one primary version over n_i successive requests, and the skip of at most one alternate over s_i successive requests
- Meet the deadline of each Black job by the primary version
- Meet the deadline of each Grey job, either by the primary version or the alternate one.
- Optimize the number of successful executions for primary versions
- Minimize the number of job'losses

A scheduling framework for the BGW model is specified by a combination of :

- a scheduling rule for the primaries,
- a scheduling rule for the alternates,
- a scheduling rule for the white jobs
- and a hierarchy between the three schedulers.

A job is in the first list only if it is a black job, in the two first lists if it a grey job and in the third list if it is a white job. Moreover, this framework is hierarchical in that sense that the first two schedulers must be organized under either the First Chance approach or the Last Chance approach. Finally, the list of white jobs has to be served when no jobs are present in the two first lists i.e. as a background scheduler. Several combinations are currently under study.

VII. Conclusion

In this paper, we describe an ongoing work on a novel QoS model that aims to improve the Quality of Service for firm real-time systems. We propose to modelize the execution requirements of periodic tasks with the Skip-over approach and the Deadline Mechanism, both integrated in the BGW model. We are validating our approach by means of simulation experiments where different scheduling frameworks are compared.

References

[1] G.C. Buttazzo, Luca Abeni. Elastic Task Model for Adaptive Rate Control, *The 19th IEEE Real-Time Systems Symposium*, pp. 286-295, 1998.

[2] M.Chetto-Silly. The EDL Server for Scheduling Periodic and Soft Aperiodic Tasks with Resource Constraints, *Real-Time Systems*, 17(1), pp 87-111, 1999.

[3] H. Chetto, M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering*, Volume 15, Issue 10, pp. 1261-1270, 1989.

[4] M. Hamdaoui, P. Ramanathan, A dynamic priority assignement technique for streams with (m, k)-firm deadlines, *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 14431451, 1995.

[5] C.-C. Han, K.G. Shin, J. Wu. A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults. *IEEE Transactions on Computers*, Vol. 52, no. 3, pp. 362-372, 2003.

[6] G. Koren, D. Shasha. Skip-over algorithms and complexity for overloaded systems that allow skips. *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, Pisa, Italy, 1995.

[7] A.L. Liestman, R.H. Campbell, A Fault-Tolerant Scheduling Problem, *IEEE Transactions on Software Engineering*, vol. 12, no. 11, pp. 1089-1095, 1986.

[8] C.-L. Liu, J.-W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. of the Association for Computing Machinery*, 20(1), pp. 46-61, 1973.

[9] A. Marchand, M. Chetto, Quality of Service Scheduling in Real-Time Systems, *International Journal of Computers, Communications and Control*, (ISSN 1841-9836), 3 (4), pp. 354-366, 2008.