Schedulers for BGW Tasks to Guarantee Quality of Service of Embedded Real-Time Systems

Maryline Chetto, Mohamed Ould Sass

Schedulers for BGW Tasks to Guarantee Quality of Service of Embedded Real-Time Systems

Keywords: Embedded, Real-time, Processor Overload, Fault-Tolerance, Uniprocessor, Scheduling.

Abstract: We present a new task model called BGW for preemptable, periodic task sets, scheduled on a uniprocessor embedded platform. The tasks may be subject to faults and the processor may be overloaded. According to BGW, any Black job has to execute a primary algorithm before deadline, any Grey job may execute either the primary or the back-up algorithm and any White job may be discarded. We describe several Earliest Deadline First (EDF) based scheduling frameworks suitable for this model. We also present and discuss the results of experiments that compare the EDF scheduler applied to conventional Liu and Layland task sets to various schedulers applied to BGW task sets. The Quality of Service is observed through metrics including ratio of deadline success, preemption rate, etc.

1 INTRODUCTION

We consider the problem of scheduling preemptable, periodic real-time task systems with arbitrary relative deadlines, scheduled on a single processor by an on-line scheduler. We focus our attention on firm real-time systems for which producing results after deadline can be accepted only under some pre-specified conditions that depend on the application.
The problem of real-time scheduling has been studied extensively from about forty years starting with the famous research paper of Liu and Layland in 1973 (0). Most of these works (see a survey in (0)) have focussed on hard real-time systems and resulted in a number of fixed and dynamic priority driven scheduling algorithms with associated off-line schedulability tests.
In this paper, we propose to describe and evaluate a new task model for answering requirements of firm real-time systems that accept deadline violations due to either occurrence of faults or/and processing overload. Overload conditions can be caused by a bad system design, not anticipated simultaneous arrivals of interrupts, hardware defects in data acquisition from sensors, under-estimated computational demands, operating system exceptions, etc. Fault-tolerance techniques intend to keep the system operational in the presence of faults, even with producing degraded results. We will show how the BGW task model permits to guarantee online graceful and controlled degradation of the Quality of Service in embedded real-time systems.

2 BACKGROUND AND RELATED WORK

2.1 Traditional task scheduling

Traditionally, a periodic task τ_i is characterized by two parameters at least: C_i, worst case execution time and T_i, activation period. Every task periodically generates an infinite set of jobs for execution. The utilization factor of a periodic task gives the ratio of execution requirement per period: $u_i = \frac{C_i}{T_i}$. As a consequence, the total utilization factor of a task set composed of n tasks is : $\sum_{i=1}^{n} u_i$. The following EDF schedulability test was established (0): $\sum_{i=1}^{n} u_i \leq 1$ i.e. total utilization is at most one. A task set is said to be feasible if there exists at least one schedule where all jobs of all tasks complete by their deadline at run time. Earliest Deadline First was proved optimal and Deadline Monotonic was proved the best one in the class of fixed priority schedulers (0). However, EDF as well as DM are suitable for under-loaded processing systems where the processing demand is lower than the processing capacity at every time. Such schedulers are particularly adapted to a hard real-time context that imposes underload conditions.

2.2 Fault-tolerant scheduling

Redundancy is the foundation of fault tolerance techniques. There are three types of redundancy: hardware, software and temporal. Permanent faults are generally dealt through hardware redundancy while temporal redundancy techniques serve for transient

or intermittent faults. They consist in re-executing a task which has failed with either the same coding version (pure temporal redundancy) or a different version, currently a shorter one. We are interested with the Deadline Mechanism (DL) model. Each task has two independent software versions (0). Firstly, a major version called primary produces results with high precision when it is completely executed before deadline. Secondly, a version called alternate or back-up with shorter execution time has to run for producing a just acceptable result whenever the primary fails in executing timely due to a fault or processor overload. Two distinct scheduling frameworks may be implemented for the DL mechanism (0) (0). Firstly, according to the First Chance (FC) technique the alternate version of any job executes completely first before the primary version of the same job starts execution. If the primary version finishes before deadline, its results are used in preference to those of the alternate. Secondly, the Last Chance (LC) technique attempts to execute first the primary version. Nevertheless sufficient processing time intervals have to be reserved to guarantee feasible execution of the alternate version if the primary fails. Consequently, success of any primary leads to discard the corresponding alternate and recover processing time since the result of the alternate becomes no longer necessary. In this strategy, the scheduler has to suspend any running primary whenever an alternate requires to be executed so as to meet its deadline. Theoretical and simulation studies established that the LC strategy outperforms the FC strategy (0) (0).

2.3 Scheduling with overload conditions

Any scheduling algorithm should aim at minimizing the overall damage to the system performance whenever a processing overload occurs. This can be performed by dynamically changing some timing parameters of the tasks (e.g. execution time or period), using importance values attached to the tasks or skipping some jobs of recurring tasks. In that work, we opted for the Skip-over (SO) model (0). Each periodic task is characterized by a value called skip factor denoted by s_i, $(2 \leq s_i \leq \infty)$ signifying that among s_i successive jobs, at most one can be skipped. Every job of a task has one of the two colours: red or blue. A red job has to complete before deadline while a blue one can be aborted at any time. Moreover, after a deadline missing, at least $(s_i - 1)$ jobs are red and must be executed timely. Several scheduling schemes have been proposed and analysed for the SO model such as RTO (Red Tasks Only) or BWP (Blue When Possible).

3 THE BGW TASK MODEL

BGW (Black Grey White) is a novel task model which uses time redundancy to cope with both transient processor overload and faults (**?**). The BGW model derives from:

- the DL Model where each periodic task has two independent versions for fault-tolerance and overload management.
- and the SO model where each periodic task has a skip parameter for overload management.

Every job generated by a periodic task take at every time instant one of the three following colours :

- *Black* if the job has to imperatively produce a result through the primary version,
- *Grey* if the job has to produce a result through at least one version, but in preference the primary one,
- *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, a BGW task set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ is composed of n periodic tasks where each task τ_i is characterized by its conventional timing parameters and two additional *QoS parameters* (n_i, l_i). Let us define the term "distance" as a number of requests. n_i expresses the maximum distance allowed between two consecutive successful executions of the primary version. l_i is the maximum distance allowed between two consecutive successful executions of a job (whatever primary or alternate).

Any scheduling scheme for BGW-tasks should :

- guarantee that the requirements of the BGW tasks are satisfied. At least one primary version over n_i successive jobs has to be executed timely, and one alternate version over l_i successive jobs has to be executed timely. In other words, the scheduler should timely execute the primary version of each Black job and either the primary version or the alternate version of each Grey job.
- maximize the number of successful primary executions,
- minimize the number of unsuccessful jobs i.e jobs which are either discarded or not completed before deadline.

In this paper, we report the results of a simulation study where the performance of three EDF based schedulers is analyzed (0).

4 SCHEDULERS UNDER STUDY

We apply the FC technique where every alternate executes entirely and systematically for producing a result with just acceptable precision. After the corresponding primary is authorized to start execution for producing a result with a better precision but nevertheless with a longer execution time. By definition of the Black colour, only the primary version is executed for black jobs. Either the alternate version or the primary one has to be executed timely for every grey job. The white job can be aborted.

The FC technique applied to the the BGW model can be implemented through different scheduling variants. We analyse here three EDF based scheduling strategies, each one defined by a specific ordering of versions. Let "$X > Y$" express that job with type X should be executed with a higher priority than any job with type Y. Any scheduling framework uses at most five ordered lists which are respectively BP (BA does not exist by definition of black colour), GA, GP, WA and WP. This can be easily implemented in any real-time operating system with only one list of jobs that contains five ordered sub-lists. We assume in that work that all the lists are ordered according to the earliest deadline first rule. Our simulation results will concern scheduling frameworks based on the following priority ordering:

1. $BP > GA > GP > WA > WP$: Both the alternate and the primary versions of any grey job should be executed with a higher priority than the white jobs. As a consequence this scheduling policy will be denoted by G*b*WA (Grey *before* White Alternate),

2. $BP > GA > GP > WP$: Both the alternate and the primary versions of any grey job should be executed with a higher priority than the primary versions of the white jobs. In that policy, alternate versions of white jobs are never executed. As a consequence this scheduling policy will be denoted by G*b*WP (Grey *before* White Primary),

3. $BP > GA > WA > GP > WP$: The alternate versions of all grey jobs and white jobs should be executed with a higher priority than the primary versions of all grey jobs and white jobs. As a consequence this scheduling policy will be denoted by A*b*P (Alternate *before* Primary)

5 EXPERIMENTS

5.1 Simulation Environment

We developed a task set generator that outputs BGW-schedulable task sets. The task generation procedure was parameterized so that task sets exhibit different degrees of scheduling difficulty and consequently allows us to provide an objective evaluation of the BGW mechanism.
The generator has the following input parameters: number of tasks (n), Least Common Multiple of the periods (P), worst case alternate load (U_a), worst case primary load (U_p), (n_i) and (l_i) parameters.
In all simulations reported in this paper, parameters n and P take constant values 22 and 3360 respectively. We considered 14 values for Up which uniformly vary from 0.8 to 2.2. U_a is a linear function of U_p with $U_a = 0.2 * U_p$ and $n_i = 7, l_i = 4$.
Outputs are the timing parameters of tasks (i.e. period, relative deadline, worst case alternate execution time and worst case primary execution time. More precisely, for a given alternate load U_a , C_i^a and C_i^p are proportional to T_i ($C_i^a \leq C_i^p$) with a minimal value equal to 1. The task sets which result from the different combinations of U_a and U_p were scheduled according to BGW and EDF successively.

5.2 Metrics

We measure the resulting Quality of Service of the system under the BGW mechanism by the ratio of primary versions which are executed timely over the total number of jobs (NPJ) and the ratio of jobs (primaries and alternates) which are executed timely over the total number of jobs (NSJ). Scheduling tasks in overloaded conditions implies to discard some uncompleted jobs. Consequently, as processor time can be wasted, we measure the wasted time ratio (WTR) i.e. the percentage of time used by the processor for producing no result or useless results.
The EDF scheduling strategy wastes time because of uncompleted primaries. Under the BGW strategy, time is wasted when both primaries and alternates are uncompleted and white jobs do not execute.There are also some processing time lost in preemptions and context switches. So we measured the relative preemption cost (RPC) i.e. the number of preemptions per the total number of jobs within a time reference window.

5.3 NSJ Analysis

We analyse NSJ which gives the number of jobs which are executed timely (either by primary or alternate versions) over the total number of jobs. We compare it for the three different BGW strategies in addition to the classical EDF scheduling algorithm (where every job has only one version i.e. the primary one).
Fig.1 shows variation of NSJ by making vary the primary load, U_p, (and consequently the alternate load U_a). As shown by the four graphs, for high load, A*b*P and G*b*WA strategies outperform G*b*WP and EDF.
The A*b*P strategy exhibits a higher NSJ in comparison to the G*b*WA strategy, which in turn, is better than G*b*WP. Indeed, under low alternate load (U_a) compared to primary load (U_p), there are additional chances of executing alternate versions of grey and white jobs, when they have a higher priority in the job execution order.
The basic EDF scheduler causes more jobs to fail compared to the three scheduling strategies applied to BGW-tasksets. Only primary versions are executed under classical EDF, thus requiring large processing time. These observations confirm the effectiveness of our specific task model for overload control. For all BGW strategies, the number of deadline misses increases as U_p increases. High value of U_a causes more processor time consumption by alternate version of grey jobs, thus leading to higher deadline misses for grey primaries. Clearly, the priority of alternate versions in the job execution order significantly impacts the BGW performance in terms of global success. This confirms usefulness of A*b*P and G*b*WA strategies.
From $U_p = 100\%$, NSJ highly decreases until reaching 50% for $U_p = 200\%$ under the EDF algorithm, and 40% under the G*b*WP strategy. For A*b*P and G*b*WA, the decreasing is insignificant and independent from the primary load, until $U_p = 210\%$ where NSJ visibly starts decreasing. A*b*P and G*b*WA have similar NSJ until U_p reaches 170% where a small difference between the two strategies can be observed.
Under higher load, the ratio of deadline misses for EDF algorithm is approximately ten times higher than for the A*b*P strategy. This observation outlines why both the BGW model and specific scheduling strategies improve significantly the resulting Quality of Service of embedded systems under transient processor overload. NSJ appears to have the highest values for the A*b*P and G*b*WA strategies, when the GA list respectively WA list has higher priority than the GP list respectively the WP list.
The G*b*WP policy behaves like EDF when the primary load (U_p) is increasing. Therefore, executing WP jobs before WA jobs leads to a significant decrease in global success for G*b*WP relatively to G*b*WA because primaries have a higher execution time in comparison to alternates.

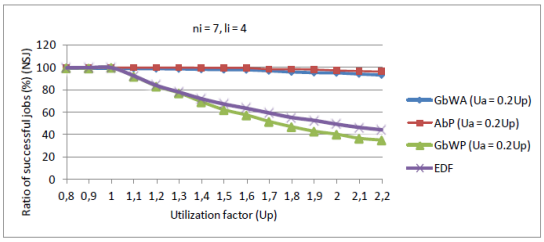

Figure 1: Percentage of successful jobs as a function of load.

Let us note that $1/(n_i+1) = 14.28\%$ and $1/(l_i+1) = 71.72\%$ that respectively represent the ratio of black jobs and grey jobs which have to meet deadlines. When the primary load Up is high, all the BGW policies execute the lowest number of black jobs which is permitted by distance parameters. In that situation, the BGW and EDF policies behave identically in terms of success ratio. Nevertheless, there is no control on which job fails under classical EDF in contrast to BGW policies.

5.4 NPJ Analysis

Fig.2 depicts variation of NPJ i.e. the ratio of successful primaries by making vary U_p (primary load). For all values of U_p, G*b*WA outperforms EDF.

G*b*WP and EDF strategies offer the best performance since, for the former strategy, the highest priorities are affected to the GP and WP jobs. Whereas, for the latter strategy, there is no execution of the two versions of grey and white jobs by re-executing the primary version after executing the alternate one.
NPJ is greater for the G*b*WP strategy, compared to the G*b*WA strategy which is in turn, greater than the A*b*P algorithm.
With the small difference observed between the EDF algorithm and the G*b*WP strategy in term of successful primaries, we can state that, the BGW strategies have a comparable performance if priorities given to GP and WP jobs are greater compared to priorities given to the WA jobs.
It is very interesting to note that NPJ, the percentage of successful primaries remains greater than or equal to 14.28%, that is the smallest ratio of primary versions to be executed in accordance with the requirement of the BGW model.

At least $1/(n_i)$ jobs have to execute their primary version. In this study we have $(n_i = 7)$. As a consequence, at least $1/7 = 14.28\%$ primaries must be executed timely. This explains why we can continue the simulation experiment until $U_p = 700\%$ and we observe that NPJ is decreasing under 14.28%. It is also observed that NPJ for EDF decrease rapidly as U_p grows, until it stands comparison to the A*b*P and G*b*WA policies.
In fact, at higher values of the utilization factor U_p, in particular between $U_p = 210\%$ and $U_p = 220\%$, we observe that all schedulers behave identically when observing NPJ i.e. the ratio of successful primaries.

Considering the jobs execution order under BGW

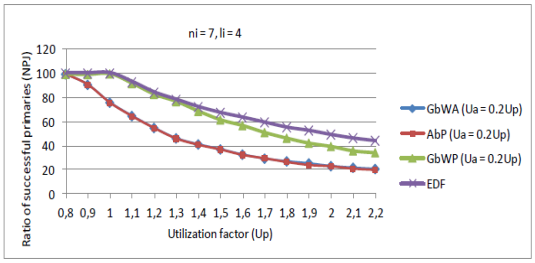

Figure 2: Percentage of successful primaries as a function of load.

scheduling strategies, we notice that when the computation times of GA jobs and WA jobs tend to zero, all BGW scheduling policies tend to behave as the EDF scheduler.
We note also that, the lower the priority is given to WA jobs, the more similar are the behaviours of BGW and EDF policies in terms of successful primaries ratio. When the primary load U_p becomes very high (more than 210%), all the BGW policies execute only primaries of the black jobs and so BGW and EDF policies have similar NPJ. We can draw the same conclusion when the alternate load is roughly equal to the primary load.

5.5 WTR Analysis

The wasted time ratio (WTR) is the percentage of time used by the processor for producing useless results (notably jobs which are aborted). The EDF scheduling strategy leads to waste processing time when primaries are aborted before completion because of time starvation. Generally, wasted time comes from the execution of primary grey jobs and all versions of white jobs in the BGW policies.
Fig.3 shows the variation of WTR.

The best performance regarding WTR is given by the G*b*WA algorithm for all load conditions. In under-load conditions, WTR is equal to zero for EDF because only the primary versions are executed, and EDF is optimal if there is no overload. For G*b*WP, WTR is also equal to zero until $U_p = 110\%$ and increases slowly until $U_p = 130\%$. Then, it continues to increase with large values until 21% at $U_p = 220\%$.

Both the increase in the number of lost jobs and

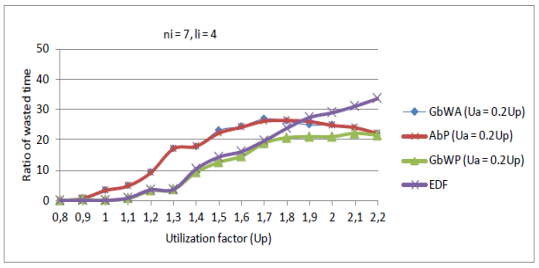

Figure 3: Percentage of wasted time as a function of load.

the increase in computation time of alternate and primary versions lead to the increase in wasted time. Indeed, abortion of any primary or alternate version will create large unusable processing time.
As U_p increases, the ratio of lost jobs under the EDF policy increases more than the A*b*P and G*b*WA strategies.
Hence, WTR under EDF is greater than under A*b*P and G*b*WA strategies when U_p is very high.

5.6 RPC Analysis

It is interesting to compare the number of preemptions generated by the different scheduling strategies, so as to evaluate correctly the relative overhead. In fact, the previous performance evaluation may have no significance if some schedulers exhibit unacceptable overhead at runtime due to context switches.
Fig.4 shows RPC i.e. the preemption ratio

In under-load conditions, we notice a light difference between the different algorithms. Globally, RPC is approximately constant at 19% for the A*b*P and G*b*WA strategies. We also observe similarity between G*b*WP and EDF policies with different values of alternate loads.
RPC for A*b*P and G*b*WA strategies is greater than for G*b*WP and EDF algorithms. Consequently, the number of selected WA, GP and WP jobs to be executed for A*b*P and G*b*WA strategies increases, which increases the preemption ratios.

Under all load conditions, the G*b*WP strategy offers the best performance in terms of RPC compared

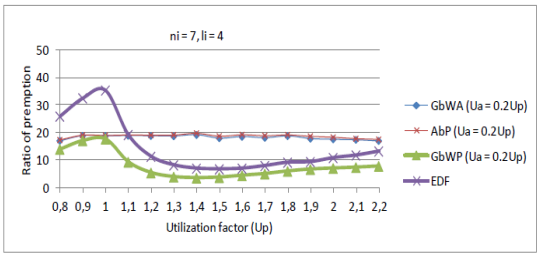

Figure 4: Preemption Rate as a function of load.

to the other BGW strategies. It is always less than 10% above $U_p = 110\%$. We observe small differences between G*b*WP and EDF policies. EDF gives high RPC in under-load conditions and decreases in overload conditions between 100% and 150%. RPC starts to increase from $U_p = 150\%$ for EDF until it reaches the same level as A*b*P and G*b*WA.
Globally, RPC is stable for A*b*P and G*b*WA policies unlike G*b*WP and EDF strategies. RPC for BGW policies, particularly for A*b*P and G*b*WA is higher than for the EDF scheduler applied to jobs which are all primary versions. When the primary version load is high, the probability to execute timely the GP jobs decreases. Thus, neither the execution of WA nor WP jobs is achieved, hence, the decrease in RPC for the G*b*WA strategy. According to jobs execution order under BGW strategies, for example for G*b*WA when the computation time of GA and WA jobs tends to be close to ones of the GP and WP jobs, the BGW policies tends to behave as EDF in terms of RPC.

6 CONCLUDING REMARKS

The contribution of this paper was twofold. We described a new approach for modelling Quality of Service (QoS) requirements of periodic task systems which may be the object of both transient faults and processor overloads. This approach consists in integrating the Deadline Mechanism and the Skip-Over model in a unified task model, namely BGW. We have shown how BGW permits to provide an acceptable quality of service through adequate task scheduling. We evaluated several scheduling schemes for BGW-task sets with experiments. The simulation demonstrates the merits of our proposed task model compared to the classical Liu and Layland task model. Simulations show the improvement of specific scheduling frameworks for the BGW model achieved in terms of ratio of successful jobs and efficiency of processor usage which alleviates the performance degradation under overload conditions.

REFERENCES

G. C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer academic, 1997.

A. K. Atlas, A. Bestavros. *Statistical rate monotonic scheduling*. In Proceedings of IEEE Real-Time Systems Symposium, December 1998.

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.

H. Chetto, M. Chetto. *An adaptive scheduling algorithm for fault-tolerant real-time system*. Software engineering journal May 1991.

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.

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

C. Liu and J. Layland. *Scheduling algorithms for multiprogramming in real-time environment*. Journal of ACM, 1(20) :46-61, October 1973.

M. Ould Sass, M. Chetto, A. Queudet. *The BGW model for QoS aware scheduling of real-time embedded systems*. MobiWac '13 Proceedings of the 11th ACM international symposium on Mobility management and wireless access. Pages 93-100 ACM New York, NY, USA 2013.