Extended Global Dual Priority Algorithm for Multiprocessor Scheduling in Hard Real-Time Systems

Josep M. Banús, Alex Arenas

Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
{josepm.banus, alexandre.arenas}@urv.net

Jesús Labarta

Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
jesus@ac.upc.edu

Abstract

In this paper we present a global scheduling method for shared memory multiprocessor systems that provides a fixed-priority preemptive scheduling of periodic tasks, hard aperiodic tasks and soft aperiodic tasks on a set of identical processors. The method is based on the functioning of the Dual Priority Scheduling Algorithm extended to work in a multiprocessor environment. This algorithm guarantees periodic tasks deadlines and achieves good mean soft aperiodic response times. We include hard aperiodic tasks, by using an acceptance control test for service. Extensive simulations show that the proposed algorithm gives both high guarantee ratios for hard aperiodic tasks still achieving low mean aperiodic response times.

1. Introduction

Traditionally multiprocessor scheduling has been treated as a particular case of multiple individual uniprocessor because of the NP-Hard nature of the problem. This is called *local scheduling* or *partitioning method*. With this methodology, one first allocates statically tasks to processors and, after that, an optimal uniprocessor scheduling algorithm is used individually on each processor. The alternative is the *global scheduling* methodology, also called *non-partitioning* method. In this case, there is a global scheduler that dynamically binds tasks to processors, obtaining dynamic load balancing, fault tolerance, etc.

In the recent years the non-partitioning method is receiving much more attention from the research community. The majority of recent works deal with utilization upper bounds for the Global Rate Monotonic Scheduling, using these bounds as a necessary schedulability condition and to perform new tasks admission control. Usually these upper bounds are too pessimistic and produce low processor utilizations. Hence, the solution to find schedulability in heavy loaded systems usually relies on their simulation. The non-existence of efficient schedulability tests, the existence of multiprocessor anomalies and its inherent computational complexity are some drawbacks of the non-partitioning method. Some of these drawbacks can be avoided using some heuristics.

Bearing in mind these problems the consideration of heterogeneous task types becomes a difficult but important issue. Usually only a single type of tasks have been considered (mainly periodic tasks). Recently some works deal with more than two types of tasks [1,2] using a static allocation scheme.

In this paper, we present an extension of the Global Dual Priority Algorithm [3] to schedule hard aperiodic tasks on shared memory multiprocessor systems (in addition to periodic and soft aperiodic tasks). We have also modified the low priority policy to reduce the number of preemptions and migrations. Our main goals are: 1) to provide a low cost acceptance test for hard aperiodic requests, 2) to guarantee deadlines for accepted hard aperiodic tasks, 3) to maintain the properties of GDP (i.e. to guarantee all periodic task deadlines and to control the delay of periodic task executions to serve aperiodic requests) and 4) provide a best-effort service to soft aperiodic tasks, trying to achieve low average response times. The capability of dynamically balancing work among processors, especially periodic tasks, will make the difference in performance with respect local scheduling algorithms.

We assume a priori knowledge of periodic tasks worst-case execution times (C_i), periods (T_i), deadlines (D_i) and arrival times ($\alpha_k = kT_i$). Aperiodic tasks arrival time are a priori unknown. All tasks are independent and can be preempted at any time. Finally, for the sake of simplicity, we assume all overheads for context switching, task scheduling, task pre-emption and migration to be zero.

2. Global Dual Priority Scheduling Overview

In this section, we provide an overview of GDP algorithm. More detailed information can be found in [3] and [4].

At *design time*, all periodic tasks are statically distributed among processors using any bin-packing algorithm. After that, promotion times of periodic tasks are computed subtracting to its deadline the WCRT achieved with a local scheduling ($D_i - C_i$).

Two priority levels are established: the Low Priority Level (LPL) and the High Priority Level (HPL).

Accordingly, every periodic task has two priorities, one in each level. The HPL order must be static to guarantee deadlines (the same used to compute WCRT). On the other hand, the priority assignment in the LPL can be arbitrary, even dynamic, because it does not compromise task deadlines.

At *run time***,** periodic task activations start with its LPL and are queued in the system Global Ready Queue (GRQ). The global scheduler (GS) selects tasks from this queue to be dispatched on any idle processor. Therefore, it is possible to execute periodic tasks on any processor during a certain time window that extends from the task activation to its promotion. This is the *dynamic phase* (see Figure 1). It provides some load balancing and reduces the number of tasks waiting for a specific processor. These tasks can advance work on other processors, making the system ready for future demands.

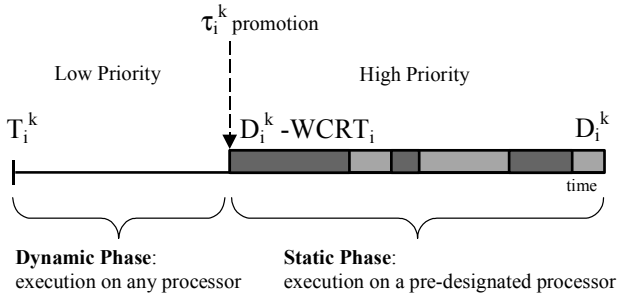

Figure 1: Periodic task τ_i^k allocation phases in EGDP

At *promotion time* of a periodic task, it must execute on its pre-designated processor. There it will only receive the interference of other higher priority promoted tasks, being its deadline guaranteed. This is the *static phase* (see Figure 1). So, when a task is promoted its priority is changed to its HPL and, when it is not running, it is moved from the GRQ to a queue local to the pre-runtime designated processor. Therefore, in addition to the GRQ, there are *m* High Priority Local Ready Queues (HPLRQ$_i$ with *i* in [1..*m*]) used to queue promoted tasks. Processors with promoted tasks are not allowed to execute tasks from the GRQ.

In the presence of aperiodic requests, periodic tasks can be delayed until their promotions to improve response time for aperiodic requests. Soft aperiodic tasks do not have deadlines (nor promotions). Because we want soft aperiodic tasks to execute before other not urgent tasks, we assign them a priority equal to zero. This implies the highest priority in the LPL. When a soft aperiodic task arrives, it is queued FIFO in a Global Ready Queue, close to the head, expecting imminent service. See Figure 3 at the end of this section 3 for more details on the run-time with heterogeneous tasks. In [3] it is shown that this scheme achieves good mean soft aperiodic response times.

3. Extended Global Dual Priority

We have improved the priority assignment in the LPL, which can be arbitrary. We propose to use an Earliest Promotion First (EPF), where the low priority refers to the periodic task promotion time instead of a fixed priority. This will accommodate to the execution of hard aperiodic tasks in EDF before their deadlines (see following paragraphs). Furthermore, the number of preemptions and migrations in the dynamic phase is reduced. In our simulation experiments we have measured savings in the number of preemptions and migrations that range from 10% to 18% if EPF is used, compared to the original proposal in [3]. Note that periodic task deadlines are still guaranteed because this depends only in HPL priority order and soft aperiodic tasks still have the highest priority in the LPL, because absolute promotions used in EPF are always greater than zero.

To incorporate hard aperiodic tasks in the model, we use the information provided by periodic task promotion times in an acceptance test. We keep track of next periodic task promotion time for every processor (NextProm$_p$). It can be implemented with a queue for every processor to queue local promotions. The number of elements in a queue is fixed and equal to the number of periodic tasks pre-allocated to the corresponding processor. Every time a periodic task finishes, the corresponding promotion is re-queued. Therefore, NextProm$_p$ simply is the arrival time of the head in the promotions queue for processor P_p.

When a hard aperiodic task is assigned to a processor, it reduces the capability to attend new requests on this processor. Therefore, we also need to keep track of assigned computing resources to already accepted hard aperiodic tasks. So, for every processor P_p there is a variable that records the last deadline assigned to a hard aperiodic task in this processor (LastDL$_p$). These variables store accumulated reservation for already assigned tasks to processors. Then, the condition to guarantee a hard aperiodic task $H^k = (\alpha_h^k, C_h^k, D_h^k)$ on a processor P_p at arrival time α_h^k is:

$$\max(\text{LastDL}_p, \alpha_h^k) + C_h^k \leq \text{NextProm}_p \qquad (1)$$

This condition guarantees the aperiodic task would not be interrupted by any promoted periodic task, Figure 2.

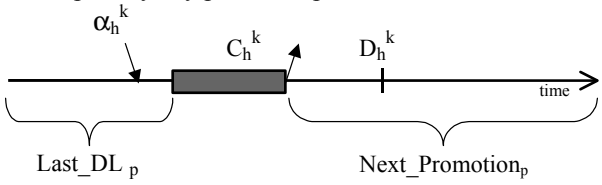

Figure 2: Hard aperiodic task time constrains for the acceptance test

Once a hard aperiodic task is assigned to a processor, its deadline is modified to execute the task with EDF. This

new deadline is what we call *effective deadline* (D'^{k}_{h}). It has to satisfy the following inequality:

$$\max(LastDL_p, \alpha_h^k) + C_h^k \le D'^{k}_{h} \le \min(D_h^k, NextProm_p) \quad (2)$$

The left hand side of inequality (2) represents the minimum possible deadline. When a task is assigned this deadline, it is executed without preemptions. The right hand side represents the maximum possible deadline, either by the restriction of the task deadline or by the promotion of a periodic task. When $NextProm_p < D_h^k$ the effective task deadline is reduced, but the initial deadline still will be met.

The *acceptance test* is as follows: (i) find a processor P_p with no promoted task and satisfying equation (1); (ii) assign an effective deadline to this task satisfying equation (2). This algorithm has a cost of *O(m)* where *m* is the number of processors in the system.

Once the effective deadline for a hard aperiodic task is assigned, the *run-time* procedure is the same as it is for periodic tasks (see Figure 3): the hard aperiodic task is queued in the GRQ with EPF priority assignment and its promotion is scheduled. Before its promotion, a hard aperiodic task can execute on any processor. After its promotion, it has to be executed on the processor where it has been assigned. It is worth mentioning that at any time there may be several hard aperiodic tasks accepted in the system.

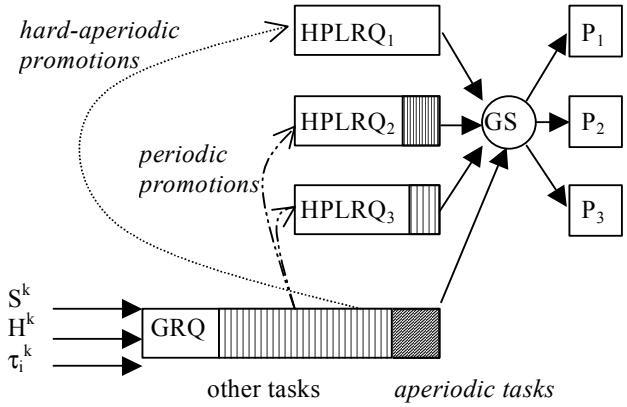

Figure 3: EGDP run-time. Squares represent processors, circles schedulers and rectangles queues. For any processor P_i , if $HPLRQ_i$ is not empty the first promoted task in P_i is executed. Otherwise P_i executes the first task from GRQ.

Balancing soft and hard aperiodic services is an important issue. It can be done allocating hard aperiodic tasks to processors and assigning them different effective deadlines. When a task is assigned a deadline greater than its minimum not promoted hard aperiodic tasks can be preempted by soft aperiodic tasks. These deadline assignments are intended to increase responsiveness to soft aperiodic requests. However, it can reduce the number of hard aperiodic tasks accepted because it increases the reservation performed for every hard aperiodic task. On the other hand, if hard aperiodic tasks are executed without the possibility of a pre-emption, soft aperiodic tasks can be delayed excessively. This balance between service for soft and hard aperiodic requests is ubiquitous in all real time systems.

A hard aperiodic request may be allocated to one of many processors satisfying inequality (1), and therefore many allocation schemes can arise. We have tested two of them: the minimum fit allocation (EGDPmin) and the maximum fit allocation (EGDPmax). The former consist on selecting the processor with the earliest promotion and assigning the minimum deadline provided by the left hand side of inequality (2), which is expected to increase the acceptance ratio and is suitable for applications with this requirement. For the maximum fit allocation, we select the processor with the latest promotion and we assign the maximum deadline derived from the right hand side of inequality (2) to the hard aperiodic request, which is expected to decrease the acceptance ratio but is suitable for applications that require soft aperiodic task responsiveness.

Another important issue is the *pre-runtime periodic tasks allocation*. The proposed hard aperiodic task acceptance test has low costs but can be very unsuccessful if periodic tasks are wrongly distributed among processors. The algorithm depends on next promotion times and these depend on the minimum period in a periodic task set assigned to a processor. The acceptance test would fail if all next promotion times were always shorter than hard aperiodic task computation requests. For example, suppose every processor has a highest priority task $\tau_1 = (C_1 = 10, T_i = D_i = 100)$. Their laxity is 90 and their promotions are 90 time units after their respective arrivals. These promotions are periodic with period 100. Hence, for every processor $NextProm_p \le 100$ and therefore no hard aperiodic task with $C_h^k > 100$ can be served.

In general, to be able to apply our algorithm the task set must satisfy the following condition:

$$C_h^k \le \max_{p \in PS}\left(\min_{\tau_{ip}}(D_i - C_i)\right) \quad (3)$$

This condition reflects the following constraint: no hard aperiodic request can be served if the processor that has the maximum spare time between periodic activations can not fit the execution time of this request.

When equation (3) is not satisfied, we propose to use a *Least Laxity First-Fit* (LLFF) pre-runtime periodic task allocation. We first sort all periodic tasks according to increasing laxity (D_i-C_i), and then we distribute tasks to processors using a first fit scheme. This concentrates as many low-laxity tasks as possible in one processor. This processor will have very short NextProm times, and the rest will have longer promotions or even could be empty (no promotions). Consequently, the acceptance test will be

more successful and will accept larger execution times for hard aperiodic tasks. We have used this allocation scheme even when equation (3) is fulfilled, although in this situation any greedy scheme that tends to leave some processors lowly loaded would succeed.

4. Results

The evaluation methodology we have used is based on the simulation of extensive randomly generated synthetic task sets. However, due to space restrictions, we are not able to provide the complete study and results. Therefore, in this section we will mainly focus on results obtained with a 65% of periodic load and 30% of aperiodic load. We have also simulated two other scheduling algorithms: a partitioning method and a global scheduler. The former was an adaptation to multiprocessors of the *Total Bandwidth Server* (TBS) [5]. Periodic tasks are allocated statically to processors but aperiodic tasks are considered globally. The global scheduler simulated was the Multiprocessor Total Bandwidth Server (M-TBS) algorithm [6] that is able to deal with heterogeneous task sets but the performance was equivalent to background service and no hard-aperiodic task was accepted (although task sets generated satisfied their conditions).

First, we tested workloads with periodic tasks and hard aperiodic tasks to check the acceptance test performance. We measured the *guarantee ratio* (GR), i.e., the number of guaranteed hard aperiodic tasks over the total number of requests. The GR achieved by EGDPmin was 90.2%, which is very high; while for TBS and EGDPmax it was about 78%. EGDPmax performance was worst than EGDPmin performance because it assigns longer deadlines to hard aperiodic tasks, i.e. it reserves more capacity for every hard aperiodic task. TBS was worse because tasks cannot migrate between processors.

Secondly, we performed experiments with all three types of tasks and we evaluated some tradeoffs between giving preference to soft aperiodic tasks or to hard aperiodic tasks. The GR achieved by EGDPmin was 99%, with TBS was 85% and with EGDPmax it was 89%. Note that GRs are higher than in the previous experiment because hard aperiodic workload now is the half. For soft aperiodic tasks, we measured the *mean average response time* (MART). The results obtained were 3.1, 1.9 and 1.07 respectively. EGDPmin had almost 100% GR but its MART was the worst (more than 3 times the requested soft aperiodic computation). The best was EGDPmax: GR=89% and MART= 1.07, both excellent. TBS had only 4% lower GR but MART was almost the double. With other workloads distributions EGDPmax and TBS performed similarly but EGDPmin's MART continuously increased as the periodic load portion decreased.

Real-time application designers can choose one of both schemes in order to achieve their goals, and they also can decide whether to assign a deadline to an aperiodic tasks or not, accordingly to its importance and to the scheme chosen. Nevertheless, there are other possibilities between EGDPmin and EGDPmax. A threshold to balance the performance for soft or hard aperiodic requests can be established in EGDP. Now, when measured MART is lower than the threshold EGDPmin is applied otherwise EGDPmax is applied. In our experiments with this threshold, as aperiodic loads became higher, we observed that MART stabilizes around the threshold. Obviously, the GR was also affected: the lower the threshold the lower the GR. We found that a threshold of 1.5 is a good compromise: soft aperiodic tasks waiting time was only a 50% of its computation time and hard aperiodic GR was above 86%. This threshold mechanism allows the application designer to automatically deal with both effects.

Finally, we evaluated the performance when hard aperiodic task's deadlines became tighter. The deadline for each hard aperiodic task was computed by applying a scaling factor to its laxity. We observed that GRs tend to decrease as deadlines became tighter. This implies that more capacity is available for soft aperiodic requests and thus the MART tends to decrease. EGDPmin and EGDPmax tended to converge but EGDPmin's GR was still higher because it is less greedy and leaves processors with more capacity. On the other hand, TBS algorithm was not able to serve any hard aperiodic task when the scaling factor was 30% or less. This is because the capacity of the server was always inferior to the requested services. This phenomenon does not happen with EGDP because it does not use any server.

5. Acknowledgement

This research is supported by MCYT project number TIN2004-07739-C02-01.

6. References

[1] Fohler, G., "Joint Scheduling of Distributed Complex Periodic and Hard Aperiodic Tasks in Statically Scheduled Systems", Real-Time Systems Symposium, December 1995.

[2] Isovic, D., Fohler, G., "Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constrains", Real-Time Systems Symposium, 2000

[3] Banús, J.M, Arenas, A., Labarta, J., "Dual Priority Algorithm to Schedule Real-Time Tasks in a Shared Memory Multiprocessor", Workshop on Parallel and Distributed Real-Time Systems, 2003

[4] Burns, A., Wellings, A.J., "Dual Priority Assignment: A Practical Method for Increasing Processor Utilization", Proceedings of the Fifth Euromicro Workshop on Real-time Systems, pp. 48-53, 1993

[5] Spuri, M., Butazzo, G.C., "Scheduling Aperiodic Tasks in Dynamic Priority Systems", Real-Time Systems Journal, vol. 10, pp. 179-210, 1996

[6] Baruah, S., Lipari, G., "A Multiprocessor Implementation of the Total Bandwidth Server", International Parallel and Distributed Processing Symposium, 2004