Matrix-DBP for (m, k)-firm Real Time Guarantee

Enrico Poggi, Ye-Qiong Song, Anis Koubaa, Zhi Wang

HAL Id: inria-00107711

https://inria.hal.science/inria-00107711

Submitted on 19 Oct 2006

Matrix-DBP For (m, k)-firm Real-Time Guarantee

Enrico POGGI*, Yeqiong SONG*, Anis KOUBAA*, Zhi WANG**

* LORIA - UHP Nancy 1 - INRIA Lorraine
2, av. de la Forêt de Haye
54516 Vandoeuvre – France
Email : song@loria.fr; akoubaa@loria.fr

** National Laboratory of Industrial Control
Technology, Zhejiang University,
Hangzhou, 310027, China
Email: wangzhi@iipc.zju.edu.cn

Abstract

(m, k)-firm means at least *m* deadlines should be met among any *k* consecutive task invocations or message transmissions. Providing (m, k)-firm guarantee is becoming attractive as it proposes an alternative between hard real-time guarantee (case of $m = k$) and soft (or probabilistic) real-time guarantee with $p = m/k$ (when $m, k \to \infty$) and allows more effective utilization of server resources (processor for task processing or bandwidth for message transmission). A dynamic priority assignment scheme called DBP (Distance Based Priority) has been proposed to handle the (m, k)-firm constraint. This paper shows that DBP combined with EDF (EDF for making choice among tasks/messages of the same priority assigned by DBP) cannot always provide good performance in a MIQSS (Multiple input queues single server) non-preemptive model. The reason is that DBP assignment is only based on the distance to failure state of each individual stream under its own (m, k)-firm constraint. It does not take into account neither the stream timing parameters (period, deadline, service time in server) nor its relationship with other streams sharing the same server. Taking into account these additional parameters, two necessary schedulability conditions are derived and an enhancement of DBP called matrix-DBP is proposed. The performance improvement has been shown by simulations.

Keywords

Real-time, (m, k)-firm, Non-preemptive scheduling, Dynamic priority assignment, Performance evaluation

1. Introduction

Traditional classification of real-time systems stands for two classes to characterize the real-time requirements of such systems: Hard Real-Time (HRT) systems and Soft Real-Time (SRT) systems.

For SRT applications, it is permitted to miss some deadlines occasionally. The term occasionally is not precise, but for SRT systems we often specify a probability to meet the deadline requirements. In general, the analysis of such systems is made using stochastic approaches and queuing theory [Koubâa02][Song02].

For applications with HRT requirements, no deadline miss is tolerated. It means that each task of a HRT application must meet its deadline; otherwise it comes to a failure. The analysis of such systems is performed with worst-case analysis to estimate an upper bound for application response time using either service curve approaches [Cruz91] or classical worst-case response time analysis [Lehoczky90].

These two classes might be insufficient to appropriately describe a real-time system. In fact, for SRT systems, stochastic analysis gives only mean response time or better the probability of deadline misses and cannot guarantee that these deadlines are missed in right manner to hold the good behavior of real-time system. An example is the MPEG video packets transmission in terms of a regular GOP (Group Of Picture) of (IBBPBBPBB) in which the packet importance is ordered decreasingly: I (Intra images), P (Predicted images), B (Bi-directional predicted/interpolated images) [Furht99]. To guarantee a certain quality a receiver should be able to receive in time at least m such packets per every k totally transmitted packets. The extent to describe how a system may tolerate missed deadlines has to be stated precisely. On the other hand, HRT systems make stringent assumptions and state that all deadlines must be met. But in practice, many systems being classified HRT are not so « hard ». Occasional deadline misses can be tolerated without necessarily leading to system failing if they are *correctly distributed* according to a *specific pattern*. For example, process control applications often give sampling (or message generating) period as deadline. But missing some of them can be tolerated [Ramamritham96]. Moreover when we consider a distributed system, taken together the worst-case response times (task execution and message transmission) and hard deadlines may be simply unfeasible for a large set of supporting system (in terms of

available computer power and network bandwidth). Another problem that a HRT may arise is that HRT guarantee is often under « good hypothesis » on the environment perturbation model. However the randomness of environment can simply lead to the HRT guarantee impossible [Navet99], [Navet00].

To resolve those problems, a new approach based on (m, k)-firm idea [Hamdoui95] called weakly-hard real-time (WHRT) has emerged to deal with real-time systems that permit some deadline misses without violating the behavior of applications. A lot of work has been done in this field to characterize the WHRT systems [Bernat97], [Bernat01] and defines a WHRT system as a system that can tolerate some degree of missed deadlines provided that this number of missed deadlines is bounded and precisely distributed. The most significant WHRT constraint is (m, k)-firm constraint and which consists of guaranteeing m out of k consecutive task executions or message transmissions, otherwise, the system is said to be in *failure state*.

New strategies to schedule systems with (m, k)-firm constraints or other similar constraints (e.g. window-constrained [West00], skip-over [Koren95]) have been defined [Bernat01], [Hamdaoui95], [Hamdaoui97], [Lindsay99], [Ramanathan99], [Quan00], [Striegel00] and the comparison criterion between these scheduling approaches is mainly the failure state probability. The simplest (m, k)-based scheduling policy is Distance Based Priority (DBP) proposed by [Hamdaoui95] which is used to schedule multiple message streams competing for service on a single server (MIQSS model) having each its own (m_i, k_i)-firm constraints. It has been shown [Hamadaoui95] that when streams have very different (m_i, k_i)-firm requirements and are identical (i.e. with the same message transmission time distribution, the same message inter-arrival distribution and the same deadline distribution), DBP is especially beneficial to tighten the failure probability based on the distance to a failure state. However, the identical streams case may be realistic for systems like video packet transmission but can be no longer true for other real time systems. Just consider the factory communication systems based on switched Ethernet [Song02], both small data packets (64 bytes) exchanges for process control and great data packet (1500 bytes) for video supervision, file downloading, etc. can co-exist. In this case, additional time-related parameters should be taken into account in

decision process when assigning the priorities. Therefore when extending DBP of Hamdaoui and Ramanathan [Hamdaoui95], which is initially designed for the identical streams case (video sources), to the general real-time system with quite different stream sources, we should take into account not only (m, k)-firm constraint but also other real-time constraints, characterized by parameters like *deadline*, *processing time*, *generating period* (or minimum inter-arrival time) and the *relative criticality* among jobs from different streams. For statement simplicity, we use hereafter the term "job" to represent either a task invocation or a message.

The main idea of the matrix-DBP is to handle both (m,k)-firm constraints and the other real-time constraints of streams. In the following, after having formally described the MIQSS model in section 2, we point out in section 3 a basic lack of DBP in assigning priority. This will give us the opportunity to state in section 4 necessary schedulability conditions that enrich the well-known limit on the workload. Section 5 presents matrix-DBP which outperforms DBP in terms of failure probability in overload scenarios. Section 6 provides some simulation results for performance evaluation of matrix-DBP. Finally in section 7 we give conclusions and point out future work.

2. MIQSS model

Multiple input queues single server (MIQSS) model can be used to study a large category of computer and telecommunication systems such as multiple tasks execution in a CPU, transmission of messages issued from multiple message stream sources sharing a same transmission medium or network interconnection equipment. The proposed model is made up of N sources generating N streams of jobs τ_i ($i = 1, 2, \ldots N$) attempting to be served by a single server.

Each stream is formed by a source and a waiting queue, where a job issued from a source waits until chosen by the server. The server chooses jobs at the head of queues according to its scheduling policy. We assume a service is non-preemptive as we mainly aim to message transmission applications. Preemptive server case has been studied in [Ramanathan99]. Notice that even in task execution context it is not

always desirable to preempt task in execution because of additional context switching overheads.
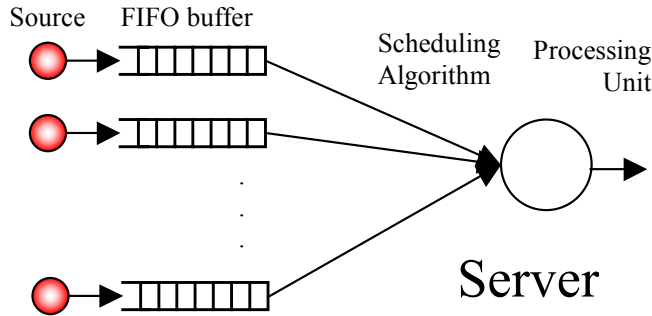

Figure 1. MIQSS model

Although streams can be periodic or aperiodic (i.e. jobs are randomly generated), we only consider the following *periodic sources*. In fact, in real-time community it is common to also consider sporadic traffic as periodic by taking the minimum inter-arrival time of jobs as period. In practice, for most of transmission systems this minimum inter-arrival time does exist (e.g. *64-bytes* packet + *96-bits* IFS in Ethernet, leaky bucket smoothed input traffic). We characterize a stream τ_i by: $\tau_i = \{T_i, D_i, c_i, m_i, k_i\}$, with *i = 1, 2,..., N* representing the index of sources, T_i the job generating period (can be message generation or task invocation period), D_i the associated deadline, c_i the job service time on the server (can be the transmission duration of a message or execution time of a task) and m_i the number of jobs meeting their deadline in k_i consecutive served jobs. We notice that we do not specify the release time (or offset) for τ_i in order to make our result more general. Moreover it is difficult to synchronize the sources in practice.

3. DBP and its drawback

DBP was firstly introduced by Hamdaoui and Ramanathan [Hamdaoui95], as a dynamic priority assignment mechanism for jobs under (m, k)-firm constraint in a MIQSS model.

The basic idea of DBP algorithm is quite simple and straightforward: the closer the stream to a *failure state* the higher its priority is. A failure state occurs when the stream's (m, k)-firm requirement is transgressed, *i.e.*, there is more than *k - m* deadline misses within the last *k-length window*.

So for each stream source τ_j, which requires an (m_j, k_j)-firm, the priority is assigned based on the number of consecutive deadline misses that leads the stream to violate its (m_j, k_j)-firm requirement. This number of deadline misses is referred to as *distance to failure state* from current state. The evaluation of this distance can be done by considering the recent history of τ_j. The key to do this is the *k-sequence*.

The k-sequence is a word of *k* bits ordered from the most recent to the oldest job in which each bit keeps memory of whether the deadline is missed (*bit = 0*) or met (*bit =1*). In this paper, the leftmost bit represents the oldest. Each new arrival job causes a shift of all the bits towards left, the leftmost exits from the word and is no longer considered, while the rightmost will be a *1* if the task has met its deadline (*i.e.* it has been served within) or a *0* otherwise. Figure 2 gives an example with (4,5)-firm constraint.
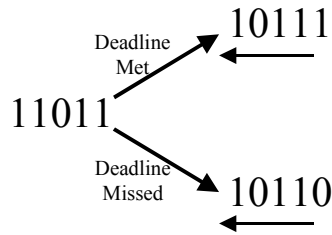

Figure 2. Possible evolution of the *k-sequence*

The priority assigned by DBP to a job at a given instant is equal to the *distance* of the current k-sequence to a failure state. This distance can be easily evaluated, by adding in the right side *0*s until failure state and the number of added *0*s is the priority. If a stream is already in failure state (i.e., less than *m* *1*s in the k-sequence), the highest priority *0* is assigned. For example, considering a stream with (3,5)-firm constraint, the current job j_{i+1} is set the priority of *2* if its previous five consecutive jobs construct the state of (11011), and is set the

priority of *3* if its previous five consecutive jobs construct the state of (10111).

Formally, according to [Hamdaoui95] priority is evaluated as follows. Let $s_j = (\delta^j_{i-k_j+1}, \cdots, \delta^j_{i-1}, \delta^j_i)$ denote the state of the previous k consecutive jobs of τ_j, $l_j(n,s)$ denote the position (from the right) of the nth meet (or *1*) in the s_j, then the priority of the $(i+1)^{th}$ job of τ_j is given by :

$$P\_DBP^j_{i+1} = k_j - l_j(m_j, s_j) + 1 \qquad (1)$$

We note that if there are less than *n* *1*s in *s*, $l_j(n,s) = k_j + 1$, so that the highest priority (*= 0*) will be assigned. This is normal as the source is in failure state.

Figure 3 shows where DBP is used for priority assignment. One of the interests of this on-line priority assignment scheme is it can be easily and efficiently implemented in hardware as each stream's history can be kept in a k_j-*bit* shift register.

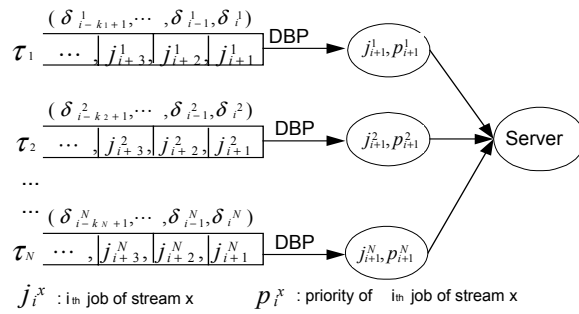j_i^x : i th job of stream x p_i^x : priority of   i th job of stream x

Figure 3. DBP for priority assignment of head-of-queue's jobs

In case of priority equality among the head-of-queue's jobs of different streams, EDF (Earliest Deadline First) is used by default.

One of the problems faced with DBP, is that it assigns priorities only considering one τ_j's (m_j, k_j)-firm constraint without comparing it to the others sharing the same server. This *self-reference* behavior may lead to a situation where more than one stream get the same

priority at the same time, in this case an algorithm to choose among them should be defined.

It is also important to underline that DBP chooses priority based on the history of the stream's k-sequence, and doesn't take into account any specific information on the actual attributes of the stream like its length c_j, its minimum inter-arrival time T_j, and its deadline D_j.

The simplest and common way to overcome these problems is to assign DBP-based priority to the jobs and, in case of priority equality, use another scheduling algorithm among the already known ones.

In their paper, Hamdaoui and Ramanathan [Hamdaoui95] combined DBP with Earlier Deadline First (EDF). However this solution gives to Deadline less importance than that given to the k-sequence, since EDF would be used only when k-sequence is not sufficient, *i.e.* when two streams get the same DBP-priority. In general, according to our earlier simulation study, using DBP with a dynamic sub-algorithm to choose in ambiguity cases may be quite disappointing. Sometimes, underestimating the information on c_j, T_j and D_j may lead to very poor results.

Consider the simple case of two streams:

Table 1. Simple Case Flow Parameter

	(m,k)-constraint	Service time (ms)	Period/ Deadline	Initial k-sequence
Sa	(4,5)	15	30	{01111}
Sb	(2,5)	2	5	{00101}

According to equation 1 the former stream has higher priority since its DBP distance is *2*, while the latter has a distance of *3*. However, transmitting the job for stream *Sa* may cause a dynamic failure state for stream *Sb* as shown in *figure 4*.

This is because during the service of a job from first stream, up to three jobs of stream *b* are generated and consequently miss their deadlines. While choosing stream *Sb* will not generate a failure state for stream *Sa*.
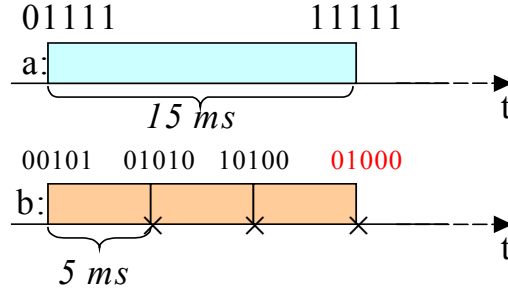
Figure 4. Worst *Sb* behaviour during service of *Sa*

This lack can be exploited in several ways, always keeping DBP as basic priority assignment function, depending on what complexity we want to introduce.

In what follows we show a specific method, which makes an average good performance without increasing drastically the complexity of the algorithm. Before doing this, we state a new necessary condition for schedulability taking into account the non-preemption and relationship between different sources.

4. Necessary schedulability conditions

Given a set of N periodic sources $\tau = (\tau_1, \tau_2, \ldots, \tau_N)$ with $\tau_i = \{T_i, D_i, c_i, m_i, k_i\}$ and with whatever release times (or offsets), the set τ is said schedulable (or feasible) if it is possible to find a scheduling algorithm allowing to meet all (m_i, k_i)-firm constraints (for $i = 1, \ldots N$). A necessary schedulability condition just means if it is not satisfied the set is surely unschedulable. But the scheduling algorithm can be very complex and finding it can be NP-hard. Whilst a sufficient schedulability condition, if it is satisfied, guarantees that the set meets its (m_i, k_i)-firm constraints with a known scheduling algorithm. The research of the sufficient schedulability condition is beyond the scope of this paper.

For a given set of N periodic sources τ, before to schedule them, it is important to have an idea on if the set is schedulable according to its (m,k)-firm constraints. We derive the following schedulablity tests for a set of (m, k)-firm streams.

Condition 1: General Schedulability

$$\sum_{i=1}^{N} \left\lceil \frac{c_i}{T_i} \frac{m_i}{k_i} \right\rceil \leq 1 \qquad\qquad (2)$$

This formula states that to have a schedulable set it is necessary, but not sufficient, to satisfy that the overall normalized workload be less than or equal to *1*. Otherwise, as the queueing time is unbounded, we are sure that the (m_i, k_i)-firm constraints will be violated and no deterministic guarantee is anyway possible. Note that the blocking factor due to non pre-emption is not considered in equation 2 as we are only stating the necessary condition.

Also, consider the case we have the same stream *Sa* of the previous paragraph and instead of *Sb* a stream *Sc* with (2,5)-firm constraint with deadline $D_c = T_c = 3$ms and service time $c_c = 1$. The equation 2 is satisfied for this case. However, even when stream *Sc* is in the farthest state from failure with k-sequence {xxx11} it cannot stand the service of any job from source *Sa*, because the number of deadline misses (at least four) is more than the (2, 5) constraints admissible (three). Figure 5 shows this situation. This suggests as *additional necessary condition* that two sources are mutually schedulable only *if the minimum number of deadlines missed by one source while serving the another one is less than the upper limit allowed by (m, k)-firm constraint.*
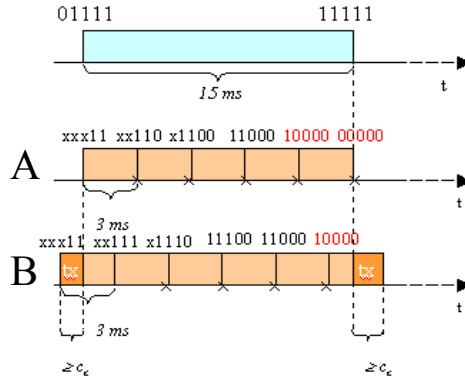

Figure 5. Possible effect of an offset between jobs of two streams

The number of deadline misses that stream *Sc* have to stand during stream *Sa* service time can change depending on the time actually left to serve the first job from stream *Sc* at the moment stream *Sa* starts its service (*offset*).

Given two streams, we are interested in evaluating the **least** number of deadline misses of one stream that would occur during the service time of another stream. For the example of Figure 5, this least number (corresponding to the best situation for stream Sc) is shown in part B, when the initial offset is equal to c_c.

More generally, for finding the least number, let us consider the best situation for a stream *Sc* during the service time of a stream *Sa* (Figure 6).
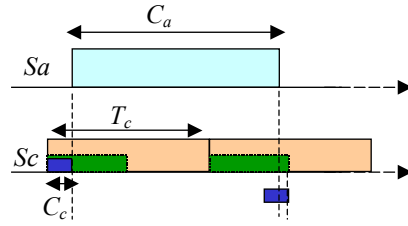

Figure 6. Best case for *Sc*: *Sa* starts its service when *Sc* terminates

Assume that during the service time of a job of stream *Sa*, *at least* one job from stream *Sc* is missed; in this case we can write (Fig.6):

$$c_a - (T_c - c_c) > D_c - c_c. \qquad (3)$$

On the other hand, the inequality that ensures us that *not more than one deadline is missed* is:

$$c_a - (T_c - c_c) \leq T_c + D_c - c_c \qquad (4)$$

Now discarding the assumption of just having one deadline miss of stream *Sc* during the service of a job form stream *Sa*, and let us call $n_{c,a}$ the *minimum number of deadline misses* for stream *Sc* during the service time of a job from stream *Sa*; for $n_{c,a} \geq 1$ we can write:

$$c_a - (T_c - c_c) > (n_{c,a} - 1)T_c + D_c - c_c \qquad (5)$$

$$c_a - (T_c - c_c) \leq n_{c,a}T_c + D_c - c_c \qquad (6)$$

Fig. 5 shows the case with $n_{c,a} = 5$.

From (5) and (6), since $n_{c,a}$ must be an integer, we conclude:

$$n_{c,a} = \left\lceil \frac{c_a + 2c_c - D_c}{T_c} \right\rceil - 1 \qquad (7)$$

This formula gives the minimum number of deadlines missed by stream Sc during a job from stream Sa is served.

What happens if $n_{c,a} = 0$? In this case the equation (7) may return either 0 or -1. This is because equation (5) evaluated for $n_{c,a} = 0$ differs from equation (3) with a term $-T_c$. However we can still use equation (7) by just adding a boundary condition that, whenever $n_{c,a} = -1$, it should be replaced by $n_{c,a} = 0$.

If we can assume that the deadline is equal to the period ($D_c = T_c$), the previous formula becomes:

$$n_c = \left\lceil \frac{c_a + 2c_c}{T_c} \right\rceil - 2 \qquad (8)$$

In the same way we can define $n_{a,c}$, $n_{a,a}$ and $n_{c,c}$, For our proposal only $n_{a,c}$, $n_{c,a}$ will be used.

Definition 1

*In a MIQSS model, two streams τ_i and τ_j are said **mutually schedulable** if $n_{i,j}$ is less than the maximum acceptable consecutive deadline misses for stream i and if $n_{j,i}$ is less than the maximum consecutive deadline misses for stream j.*

Depending on the possible temporal constraint definitions in [Bernat01] (e.g. (m, k)-firm or $\langle \overline{m}, \overline{k} \rangle = \langle \overline{m} \rangle$) this statement can be expressed by different formulas.

In the following we only interest in (m, k)-firm constrained streams. According to (m, k)-firm definition we have:

Condition 2: Mutual Schedulability

$$n_{i,j} \leq k_i - m_i$$

$$n_{j,i} \leq k_j - m_j$$

In order to extend this condition from two streams to a system with N streams, we give the following definition.

Definition 2

N streams are mutually schedulable if each couple of streams is mutually schedulable.

This necessary condition can be used together with the condition on the workload to have a more restrictive necessary condition, since even a system with load less than *1* can be found non schedulable according to the proposed criterion.

Theorem

A set of N streams is not schedulable if it does not satisfy the mutual or general schedulability test.

As a proof, it is sufficient to evaluate the load for the system made up of streams *Sa* and *Sc* of the example given in Figure 5 and note that in this case this system is not schedulable.

5. Matrix-DBP

In previous sections we arrived to point out the self-reference problem of DBP for dealing with heterogeneous streams and the need to take into account the relative criticality between streams as they

share a same server. However how to correct these DBP's lacks is not trivial. In this section, we propose to enhance the DBP priority assignment scheme by exploiting the minimum number of missed deadlines that might occur during the service times of the concurrent jobs from the other streams.

5.1 – Mutuality matrix

The heart of the algorithm we propose is a *static matrix*, called *Mutuality Matrix*, that must be updated each time the server admits a new stream or closes the connection of an old one. In general the matrix should be updated whenever a stream character is changed (e.g., service time, deadline or period changes). However, only the line and the row associated to the stream where occurred changes have to be updated, which makes the most part of these updates very fast.

The generic element of the matrix $m_{i,j}$ is the *minimum consecutive number of deadlines the ith stream can miss while stream j is being served*. Our purpose is to enhance the priority assignment scheme of DBP using the minimum number of deadline misses that may occur, so a lower bound to the actual number of deadline misses is adopted. Using the same reasoning carried out in previous paragraph and bearing in mind that all negative values ($m_{i,j}=-1$) must be shifted to zero, equations (9) and (10) give the value of matrix-elements.

$$m_{i,j} = \max(0, \left\lceil \frac{c_j + 2c_i - D_i}{T_i} \right\rceil - 1) \qquad (9)$$

And, under the hypothesis of $D_i = T_i$:

$$m_{i,j} = \max(0, \left\lceil \frac{c_j + 2c_i}{T_i} \right\rceil - 2) \qquad (10)$$

5.2 – Matrix-DBP priority

Now, each time the server has to choose between two or more concurrent streams' jobs, it updates their DBP priority by subtracting from DBP distance the corresponding matrix element. Otherwise DBP

priority is used. We note that in our proposal this DBP priority update is done dynamically rather than statically. In fact, only the DBP priorities of the *concurrent jobs* (i.e., head-of-queue's jobs) are updated. After this DBP priority update the concurrent jobs are with the matrix-DBP priorities. In case of matrix-DBP priority equality EDF is used by default.

Let Boolean variable $Q_j(t)$ denote whether the input queue of stream τ_j is empty or not at time t. $Q_j(t) = 1$ when the queue is not empty. The vector $(Q_1(t),\ Q_2(t),\ \ldots,\ Q_N(t))$ can than be used to know the concurrent jobs at time t. Formally, the priority of the $(i+1)^{th}$ job of τ_j is given by :

$$P\_MatrixDBP^j_{i+1} = P\_DBP^j_{i+1} - \max_{k=1,\ldots N}(m_{j,k}Q_k(t)) \qquad (11)$$

where $P\_DBP^j_{i+1}$ is given by equation (1).

Assuming we are checking source *Sa* and source *Sb*, defined in the last section. In this simple case, the matrix we have is the following:

$$M = \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}$$

As we said above DBP_a is *2* whilst DBP_b is *3*, but these distances have to be updated by subtracting the corresponding mutual matrix element from it as the following:

$$P\_MatrixDBP_a = P\_DBP_a - m_{a,b} = 2$$

$$P\_MatrixDBP_b = P\_DBP_b - m_{b,a} = 1$$

With this correction, higher priority is assigned to source *Sb* rather than *Sa*.

Note that $m_{b,a}$ is a lower bound of the consecutive deadline misses that can occur while a job of *Sa* is in service, it actually can be $m_{b,a}$ or $m_{b,a}+1$.

More generally because of our initial problem setting: non-preemption and unspecified relative offsets (or release times) between

streams, it is unfortunately impossible to find the actual value of a matrix element. So the correction of DBP priority by the matrix will not always result in performance improvement as can be seen in the following. By simulations we have found that sometimes using $m_{i,j}$ + 1 instead of $m_{i,j}$ can produce better performance. However how to conjunctionally use both $m_{i,j}$ and $m_{i,j}$ + 1 is still an open problem.

6. Performance evaluation

To evaluate the performance of matrix-DBP we proposed and to compare it with DBP, following scenario is simulated.

Table 2. Simulation Workload

	(m,k)-constraints	Job Process time	Period/Deadline
Stream 0	(2,5)	8/c	12
Stream 1	(4,5)	10/c	20
Stream 2	(3,6)	2/c	5
Stream 3	(1,5)	4/c	6

For this scenario, the processing power of the server c varies from *1.00* to *1.50*. The case of c = *1* corresponds to a total workload of *1* according to the general schedulability test (see equation 2). The mutual matrix with c = *1* is:

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

So the mutual schedulability test is also satisfied as in Table 2 any k – $m \geq 1$. This matrix becomes for the first time all zero when c = *1.5* ($m_{3,1}$ is the last element to become zero according to equation 10). The

simulation for c > *1.5* gives the same performance for matrix-DBP and DBP.

Whenever DBP or Matrix-DBP cannot choose by itself, EDF is used by default.

Failure state rate as well as deadline miss rate are evaluated for assessing the global performance and per-stream performance.

6.1 – Global performance

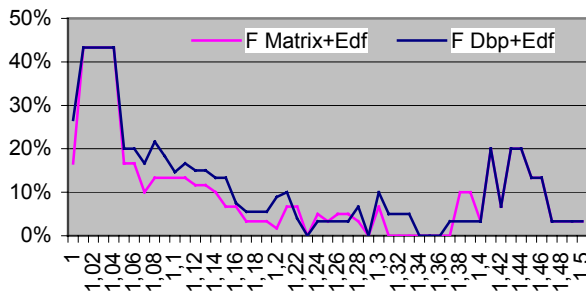Figure 7 shows the comparison for the whole system in terms of *failure states* between DBP and matrix-DBP.

Figure 7. Failure States percentage

The number of failure states is evaluated. Each point of the graphic represents the number of failure states divided by the number of jobs generated by all the stream sources during the simulation, for varying processing power of the server (*c* goes from *1* to *1.5*).

It can be seen that matrix-DBP produces less failure state than DBP in average. Note that matrix-DBP satisfies (*m*, *k*)-firm constraint for *c* within [*1.31, 1.37*] while the basic DBP only for *c* within [*1.34, 1.36*]. However, there are server power ranges for which matrix-DBP does not give the better performance than that of DBP. This is because of the inaccurate estimation of the matrix elements $m_{i,j}$ in the matrix-DBP (as explained in section 5, due to the non-preemption, it could actually be $m_{i,j}$ or $m_{i,j} +1$). In dynamic priority assignment the aim, until an exhaustive mathematical approach won't be given, is to find a scheduling algorithm that in average works better than the others. A

good idea of the behavior of matrix-DBP is given by the two curves shown in figure 8: they are plotted in polynomial fitting (fourth order of the curve of Fig. 7).
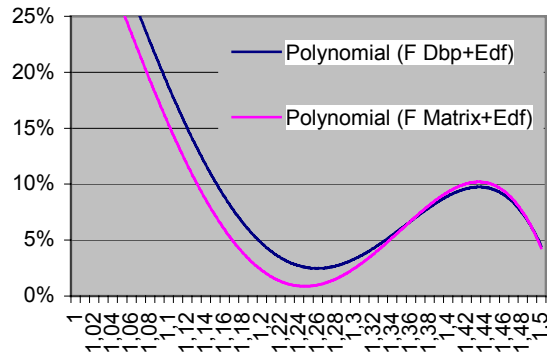

Figure 8. Polynomial representation of Failure States

Fig. 8 shows clearly the non-monotonic behavior of non-preemptive systems in terms of the failure states percentage vs. system total load.

As *c* grows the elements of the mutuality matrix approach to zero (The matrix becomes all zero for the first time when $c = 1.5$). It means that there will be a server capacity over which the behavior of both algorithms is exactly the same.

Figure 9 shows the comparison between DBP and matrix-DBP in terms of deadline misses percentage.
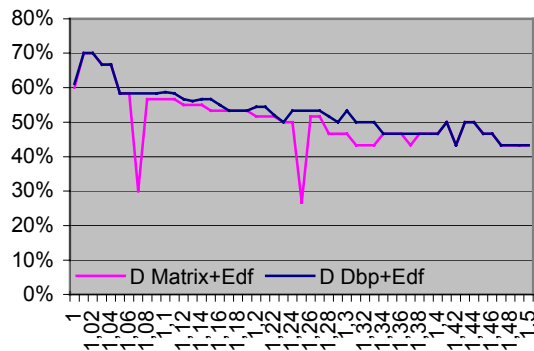
Figure 9. Deadline misses percentage

It shows that using matrix-DBP leads to a deadline misses percentage never higher than that of DBP.

The fact that the failure state percentage (Fig. 7) is not a direct consequence of the deadline misses percentage (Fig. 9) can be easily understood. As we explained in our introduction, (m, k)-firm system is different from the SRT one with m/k deadline misses percentage. Since how missed deadlines are distributed in the k-sequence is also token into account in the (m, k)-firm system.

Fig. 7 to 9 only show the global performance improvement of Matrix-DBP (point of view that interests the system designer). To understand how this improvement is achieved, let us examine the individual performance changes for each stream (point of view that interests end-users).

6.2 – Per-stream performance

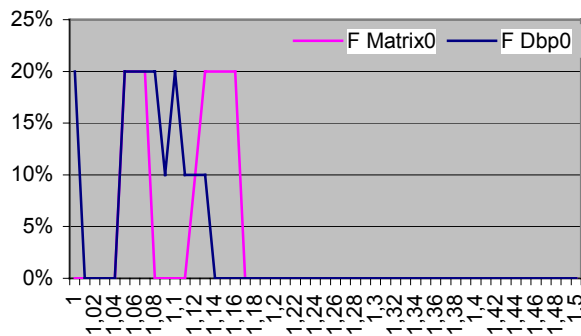Figures 10.1 and 10.2 show the behavior of DBP and matrix-DBP only for stream 0 of table 2.

Figure 10.1. Failure States for Stream 0

As the constraint is (2, 5)-firm, we can tolerate in average until *3/5 = 60%* of deadline misses. However this deadline misses may cause a failure state, if their distribution does not fit the (2,5)-firm constraints. This explains why even if deadline misses is under *60%* for some values of c from $c = 1$ to $c = 1.20$, the corresponding failure state percentages are not 0%. The failure states percentage turns to *0%* from $c = 1.17$.
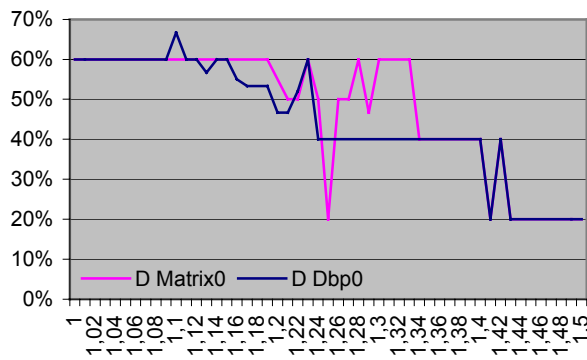
Figure 10.2. Deadline misses for Stream 0

It is also important to understand that the fact that stream 0 with matrix-DBP misses more deadlines than with DBP from $c = 1.17$ to $c = 1.35$, but without turning into failure state, is a very positive fact: matrix-DBP makes stream 0 transmit exactly the minimum number of packets it needs to keep in the success state, with the correct deadline miss distribution; this way it spares server resources to other streams.

In similar way, Figures 11 to 13 show the behavior of DBP and matrix-DBP for respectively the streams 1, 2 and 3.
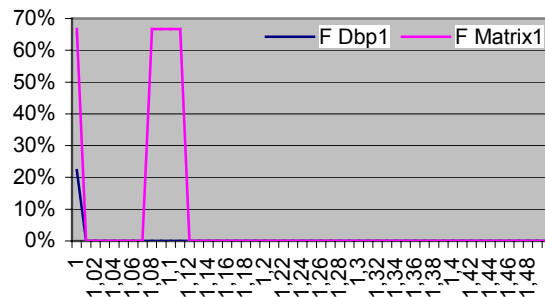

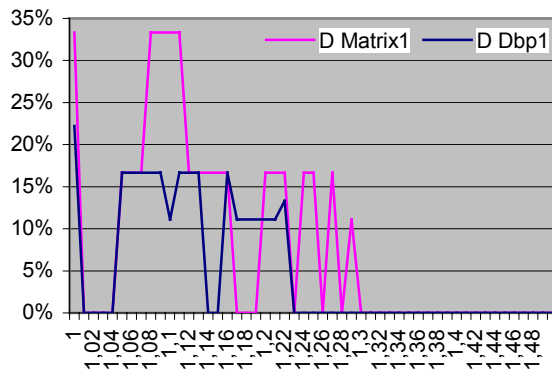**Figure 11.1. Failure States for Stream 1**

Figure 11.2. Deadline misses for Stream 1

For stream 1, *DBP* performs better than *Matrix-DBP*, but this is not astonishing: the scheduling algorithm chooses to distribute its resource among the other streams, as it should be clear from the analysis of the other graphs. Moreover, even if the failure state percentile is very high, it depends on the fact that the period is made up by few jobs (*i.e.* it is short), and not on the fact that dropped jobs are much more than in the case of *DBP* (Failure states increase of 66.6% whilst deadline misses only of 16.6%).
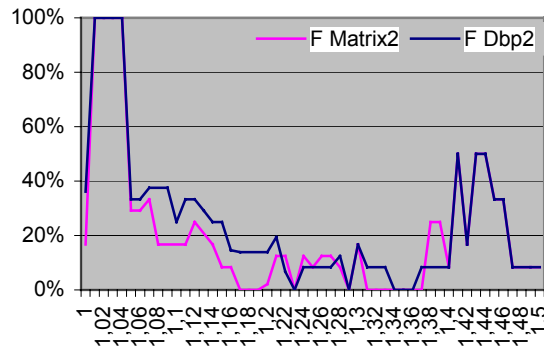

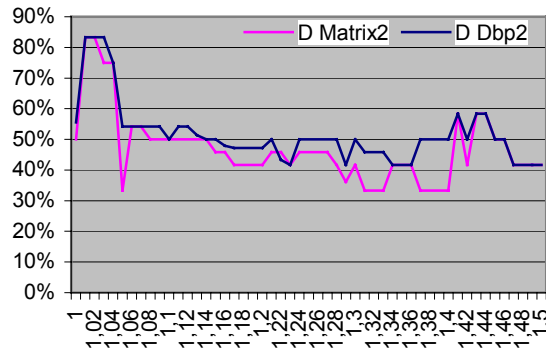**Figure 12.1. Failure States for Stream 2**

Figure 12.2. Deadline misses for Stream 2

Stream 2 has a (3,6)-firm constraint. Missed Deadline rates are under 50% but, as we pointed out when introducing the concept of (m, k)-firm, it is not enough to maintain the deadline miss rate under a given threshold. In this case, even if the deadline miss rate is lower than 3/6 = 50%, the deadline miss distribution is not met for (3,6)-firm constraint, that is why it experiences failure states (Fig. 12.3). However, we can note that number of deadline misses and failure states are kept lower for Matrix-DBP than for DBP.
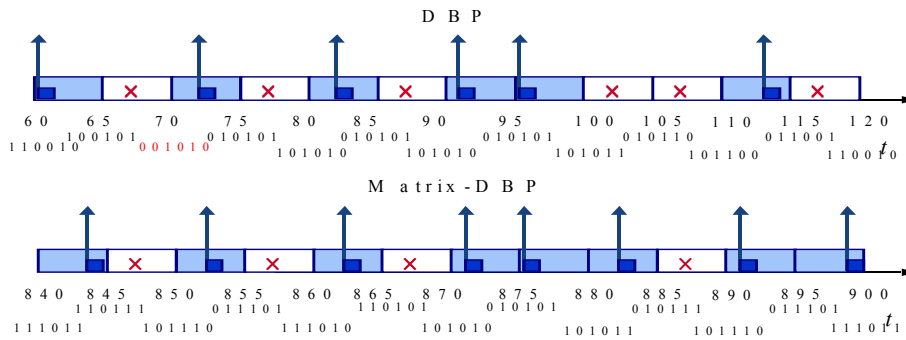

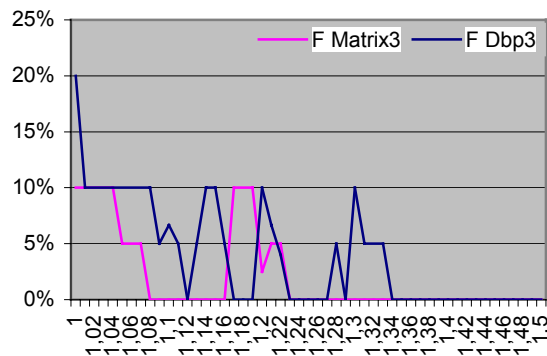**Figure 12.3. Deadline misses distribution for Stream 2 (c = 1,37)**

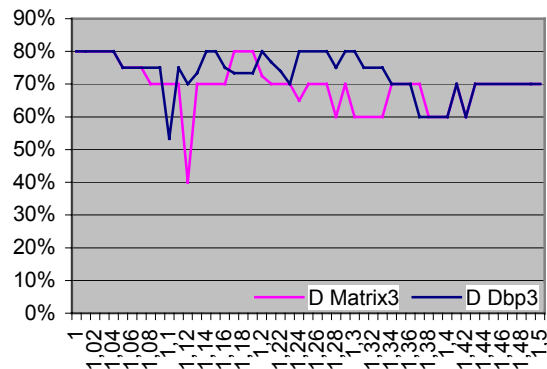**Figure 13.1. Failure States for Stream 3**

Figure 13.2. Deadline misses for Stream 3

Globally, we notice that matrix-DBP has trend to worsen the deadline meeting of stream 0 and 1 and favors the stream 2 and 3. This phenomena can be easily understood as favoring small jobs leads to decrease more efficiently the failure state numbers (or percentage) as matrix-DBP aims to decrease this parameter. For the same reason, one can notice that Matrix-DBP is not fair in terms of failure state percentage per stream.

7. Conclusion

The main original contributions of this paper are:

- Pointed out the drawback of DBP when it is applied to a more general real-time context
- Provided an additional necessary condition call mutual schedulability test
- Proposed matrix-DBP to correct DBP by subtracting from it the number of deadlines a stream is going to miss in the current situation; the proposed algorithm assumes this number is the minimum possible.

Matrix-DBP makes possible to take into account *deadline*, *inter-arrival time*, *service time* and *relative criticality* in the priority assignment scheme. Comparing to earlier solutions such as DBP+EDF which also uses deadline, in matrix-DBP these parameters are directly used within the priority assignment scheme. This means that the history of the stream is no longer more important than the information on the actual timing requirements of the stream, but the two elements are considered with the same weight in the priority assignment scheme.

More deeply, the solution of adding an already known scheduling algorithm to DBP meant to use two different decision processes, which is the reason that leads to give different importance to the two sets of information: these two processes cannot be used in parallel ! The idea we proposed, instead, merges the information that DBP ignores.

Simulations showed that matrix-DBP always outperforms DBP in terms of the global deadline miss ratio (Figure 9) and reduces in average the global failure state probability when the server is heavily loaded (Figures 7 and 8).

This improvement is with a very low computing cost or complexity since it is done by checking elements of a static matrix. This matrix, in fact, needs to be updated only when data regarding a stream source are changed or a new one is added. Even in this case it needs to update only the line and the row associated with that stream. In this sense, the implementation of our algorithm in an admission

control mechanism for providing (m, k)-firm guarantee in a network should be interesting.

One of our on going work is the extension of matrix-DBP towards multi-hop case [Lindsay99], [West00] by considering an additional parameter: the total end to end deadline.

References

[Bernat01] G. Bernat, A. Burns and A. Llamosi, "Weakly-hard real-time systems", *IEEE Transactions on Computer*s, 50(4), pp.308-321, April 2001.

[Bernat97] G. Bernat and A. Burns, "Combining (n, m)-hard deadlines and dual priority scheduling", *Proceedings of Real-Time Systems Symposium*, pages 46–57, Dec 1997.

[Cruz91] R. L. Cruz, "A calculus for network delay, Part I". *IEEE Transactions on Information Theory*, 37(1):114-131, Jan. 1991.

[Furht99] B. Furht (Editor), *Handbook of multimedia computing*, CRC Press LLC, 1999.

[Hamdaoui95] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines", *IEEE Transactions on Computer*s, 44(4), 1443–1451, Dec.1995.

[Hamdaoui97] M. Hamdaoui and P. Ramanathan, "Evaluating Dynamic Failure Probability for Streams with (m, k)-firm Deadline", *IEEE Transactions on Computer*s, 46(12), pp.1325–1337, Dec.1997

[Koren95] G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allows skips", *Proceedings of Real-Time Systems Symposiu*m, pages 110–117, Dec. 1995.

[Koubaa02] Koubâa A. and Y.Q. Song, « Evaluation de performances d'Ethernet commuté pour des applications temps réel » *Proceedings RTS'2002* Paris (France) 26-28 Mars 2002.

[Lehoczky90] Lehoczky, J.P., "Fixed priority scheduling of periodic task sets with arbitrary deadlines", *Proc. of IEEE Real-time systems symposium, IEEE Computer Press*, pp.201-209, Los Alamitos, CA (USA), 1990.

[Lindsay99] W.Lindsay and P. Ramanathan, "DBP-M, A Technique for Meeting end-to-end (m, k)-firm Guarantee requirements in point-to-point networks", *Proceedings of IEEE Conference on Local networks,* pp.294-303*,* 1999.

[Navet99] N. Navet, Y.Q. Song, "Reliability improvement of the dual-priority protocol under unreliable transmission", *Control engineering practice*, 7 (1999) pp975-981.

[Navet00] N. Navet, Y.Q. Song, F. Simonot, "Worst-case deadline failure probability in real-time applications distributed over CAN (controller area network)", *Journal of systems architecture - the EUROMICRO Journal*, 46 (2000) pp607-617.

[Quan00] G. Quan and X. Hu, "Enhanced Fixed-priority Scheduling with (m, k)-firm Guarantee", *Proc. Of 21st IEEE Real-Time Systems Symposium*, , pp.79-88, Orlando, Florida, (USA), November 27-30, 2000.

[Ramamritham96] Ramamritham, K., "Where do time constraints come from and where do they go?", *International Journal of Database Management*, 7:2, 1996.

[Ramanathan99] P. Ramanathan, "Overload management in Real-Time control applications using (m, k)-firm guarantee". *IEEE Transactions on Parallel and Distributed System*s, 10(6):549–559, Jun 1999

[Song02] Song, Y.Q., A. Koubâa and F. Simonot, "Switched Ethernet for real-time industrial communication: Modelling and message Buffering delay evaluation", *4th IEEE WFCS 2002*, Vasteras (Sweden), 27-30 August 2002.

[Striegel00] A. Striegel, G. Manimaran, "Best-effort Scheduling of (m,k)-firm Real-time Streams in Multihop Networks", *International Workshop of Parallel and Distributed Real-Time Systems (WPDRTS2000)*, Cancun (Mexico), May 1-2, 2000.

[West00] R. West and C. Poellabauer, "Analysis of a Window-Constrained Scheduler for Real-Time and Best-Effort Packet Streams", *Proc. of 21st IEEE Real-Time Systems Symposium*, Orlando, Florida, (USA), November 27-30, 2000.