Faisabilité d'une application de supervision : le projet IRMA

26 février 2003

J. Legrand, F. Singhoff, L. Nana, L. Marcé
{jlegrand, singhoff, nana, marce}@univ-brest.fr
0298016211
LIMI/EA 22 15
Université de Brest
20, av Le Gorgeu
29285 Brest Cedex

F. Dupont, H. Hafidi
{Francois.Dupont, Hicham.Hafidi}@tni-valiosys.fr
0298052744
TNI-Valiosys
Technopole Brest Iroise
Z.I. Pointe du Diable, BP 70801
29608 Brest Cedex

Résumé

Cet article présente les objectifs du projet IRMA. Dans ce projet, nous nous intéressons aux applications de supervision présentes dans les systèmes temps réel de grande taille. Par applications de supervision, nous entendons les applications qui collectent des informations de panne des équipements du système pour les analyser et finalement les présenter à l'opérateur lors des phases de maintenance. Ces applications sont constituées de tâches périodiques et de traitements activés aléatoirement lorsque des informations de panne sont délivrées au service de supervision. Nous supposons que ces deux familles de tâches partagent des ressources, et en particulier des tampons.

Dans le projet IRMA, nous cherchons à établir la faisabilité de telles applications. En première approche, nous supposons que les informations de panne sont délivrées au système de supervision de façon périodique. Dans cet article, une application est faisable si les contraintes temporelles des tâches sont respectées et si l'absence de débordement des tampons est garantie. Nous rappellons d'abord les principaux tests de faisabilité dans le cadre d'ordonnanceurs préemptifs à priorités fixes. Puis, nous proposons des bornes sur les tampons partagés par les tâches. Nos propositions sont implantées dans un démonstrateur : le simulateur Cheddar. Elles sont illustrées par une étude de cas.

Mots clefs

Faisabilité, dimensionnement de tampon, ordonnancement à priorités fixes, pannes, maintenance.

1 Introduction

Cet article présente les objectifs du projet IRMA. Dans ce projet, nous nous intéressons aux applications de supervision présentes dans les systèmes temps réel de grande taille. Par application de supervision, nous entendons les applications qui consistent à collecter les pannes et anomalies du système supervisé pour les analyser et finalement les présenter à l'opérateur lors des phases de maintenance du système.

La croissance de la complexité des systèmes temps réel fait de leur maintenance un point crucial. En effet, l'augmentation de leur taille en nombre d'équipements, leurs interactions et les multiples compétences des opérateurs nécessaires rendent coûteuses ces phases d'entretien. Souvent, un système temps réel de grande taille nécessite des périodes de maintenance régulières compte tenu du nombre élevé de composants et des pannes susceptibles d'intervenir.

Afin de diminuer les coûts de maintenance, ces systèmes proposent de plus en plus souvent des services de supervision. Aujourd'hui, les services de ce type intéressent principalement les industries aéronautiques [Inc98, Bur00] et automobiles [Div] mais de nombreux autres domaines applicatifs peuvent être concernés. Ces services aident les opérateurs à détecter et à diagnostiquer les pannes ou anomalies du système. Ils ont pour fonction de collecter ces données pendant le fonctionnement du système, de les analyser, de les traiter puis de les mémoriser. Finalement, les informations sont restituées lors des phases de maintenance du système.

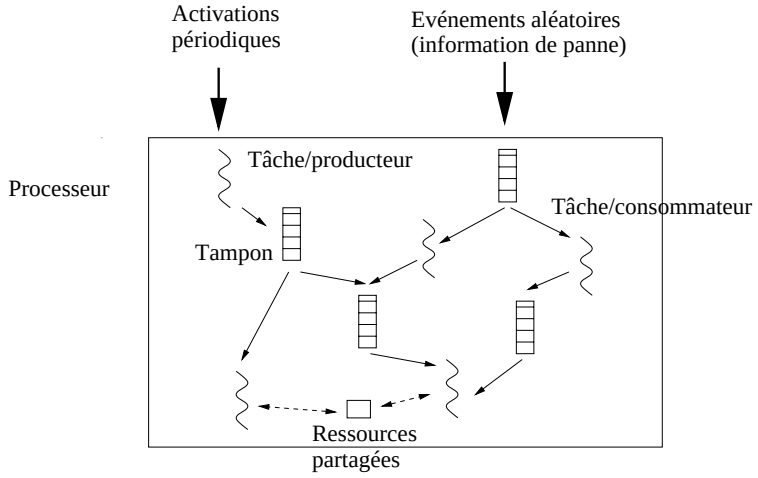

FIG. 1 – Une application de supervision

Les applications de supervision qui nous intéressent sont constituées d'un ensemble de tâches soumises à des contraintes de temps (cf. figure 1). Ces différentes tâches sont ordonnancées selon un algorithme préemptif à priorités fixes. Une partie de ces tâches est activée périodiquement mais d'autres le sont de façon aléatoire (lors des occurrences de panne dans le système). Les deux familles de tâches partagent des ressources. En particulier, les tâches produisent et consomment des informations dans des tampons de taille fixe. Les tampons et les tâches sont organisés en flot de données.

Producteurs et consommateurs sont indépendants. Ils fonctionnent de façon asynchrone : les tâches produisent/consomment à leur rythme. Ainsi, un consommateur peut se réveiller périodiquement sans pour autant disposer d'un message dans son tampon. Les consommateurs se comportent comme des serveurs qui traitent des requêtes au fur et à mesure de leur arrivée. Un consommateur réveillé alors que son tampon est vide se termine immédiatement. On suppose qu'une tâche qui produit ou consomme un message dans un tampon, effectue cette opération une fois à chacune de ses activations périodiques.

Dans IRMA, nous étudions la faisabilité de telles applications. Nous cherchons à prédire le respect des contraintes temporelles des tâches. Nous cherchons, en outre, à étudier l'occupation des tampons ;

et lorsque cela est possible, à garantir l'absence de leur débordement.

Une des difficultés réside dans l'accès aux resources partagées et aux tampons par des tâches activées périodiquement et des tâches déclenchées sur des événements dont l'occurrence est aléatoire.

Cet article relate la première phase du projet où, **en première approche, nous supposons que les informations de panne sont délivrées au système de façon périodique.** Toutes les tâches sont donc activées périodiquement.

Nous proposons des bornes sur la taille des tampons qui sont valides quel que soit l'algorithme d'ordonnancement utilisé. En effet, aucune hypothèse sur l'ordre d'exécution des tâches n'est faite lors de la construction de ces bornes.

Ces bornes sont valides pour peu que les échéances soient respectées ; et ce, même si l'ordonnancement qui sera effectivement réalisé pendant le fonctionnement de l'application est différent de celui prévu lors de la spécification du système (à cause des changements de priorité dus au partage de ressources [SRL90], des variations sur les réveils des tâches, etc).

Dans la deuxième partie de cet article, nous rappelons les principaux résultats sur la faisabilité d'un jeu de tâches périodiques indépendantes, dans le cadre d'ordonnanceurs préemptifs à priorités fixes.

Puis, dans la troisième partie, nous abordons le problème du dimensionnement des tampons manipulés par des tâches périodiques.

Ces propositions sont illustrées dans la quatrième partie grâce à une étude de cas et à un démonstrateur. Dans cette étude de cas, nous montrons que l'absence d'hypothèse sur l'ordre d'exécution des tâches lors de la construction des bornes de tampon permet de simplifier l'analyse d'une application en étudiant séparément le respect des contraintes temporelles des tâches et le débordement des tampons.

Enfin, nous concluons et dressons les perspectives de ce travail dans la partie cinq.

2 Principaux résultats concernant la faisabilité d'un jeu de tâches

On se place dans le contexte de tâches périodiques indépendantes ordonnancées par un algorithme préemptif à priorités fixes. Il existe dans la littérature de nombreux tests de faisabilité hors ligne. Ces tests sont réalisés avant exécution de l'application à partir des paramètres des tâches. Dans un premier temps, nous rappelons le modèle de tâche périodique. Ensuite nous décrivons les méthodes classiques de test de faisabilité.

Les applications ciblées sont constituées d'un ensemble de traitements répétitifs. Chacun de ces traitements est modélisé par une tâche i qui est réveillée toutes les P_i unités de temps. Ce délai fixe entre chaque réveil est communément appelé période.

Une activation remarquable d'une tâche périodique est l'instant de sa première activation. Cet instant, noté S_i, correspond à l'arrivée de la tâche dans le système. Dans cet article, nous supposons que toutes les tâches arrivent de façon synchrone dans le système ; on suppose donc que $\forall i : S_i = 0$.

En pratique, il est difficile de réaliser une activation qui soit strictement périodique. Aussi, un paramètre supplémentaire, noté J_i, spécifie un délai borné entre l'activation théorique d'une tâche et son activation réelle. Ainsi, la $k$ième activation d'une tâche périodique intervient entre les instants $k.P_i$ et $J_i + k.P_i$.

A chaque activation, une tâche exécute un traitement dont la durée est bornée par la capacité. Nous notons cette capacité C_i. Ce traitement doit être achevé au plus tard à un instant désigné sous le terme d'échéance. L'échéance est définie relativement à la période. Le délai qui sépare l'instant d'activation de l'échéance est appelé délai critique. Il est généralement noté D_i. Enfin, une tâche périodique peut être caractérisée par son temps de réponse. Généralement noté r_i, ce paramètre est le délai entre le début d'une activation et l'instant de sa terminaison. Une tâche i respecte donc sa contrainte de temps si pour toutes ses activations $r_i \leq D_i$.

A partir de ce modèle de tâche périodique, il est possible de vérifier le respect des contraintes temporelles. Une application qui respecte ses contraintes temporelles est dite faisable. Dans cette partie, nous présentons quelques résultats classiques pour un ensemble de tâches indépendantes [Riv98, Leb98, CDKM00].

Il existe trois types de tests : la simulation sur la période dite "d'étude", le test sur la charge processeur et le calcul du temps de réponse.

La période d'étude est un intervalle de temps au bout duquel un ordonnancement de tâches périodiques se reproduit de la même façon. Si les contraintes temporelles des tâches sont respectées dans la période d'étude, alors les contraintes temporelles le seront durant toute la vie du système [LM80]. Pour les systèmes considérés, la période d'étude est définie par l'intervalle de temps :

$$[0, max(\forall i : S_i) + 2.PPCM(\forall i : P_i)] \tag{1}$$

où $PPCM(\forall i : P_i)$ est la fonction qui calcule le plus petit commun multiple des périodes de toutes les tâches. La simulation est rapidement difficile à exploiter pour les systèmes comprenant des tâches dont la période implique un PPCM élevé.

Un deuxième procédé, moins sensible au passage à l'échelle, consiste à s'assurer que la charge processeur des tâches est inférieure ou égale à une borne donnée. Ainsi, pour l'algorithme Rate Monotonic, Liu et Layland ont démontré qu'une charge processeur inférieure ou égale à $n(2^{1/n} - 1)$ était une condition suffisante (mais non nécessaire) pour tester la faisabilité d'un jeu de n tâches [LL73]. Avec cet algorithme d'affectation des priorités, un ensemble de n tâches respecte donc ses contraintes temporelles si :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{1/n} - 1) \tag{2}$$

Ce deuxième test de faisabilité ne permet pas d'étudier le respect des contraintes temporelles de chaque tâche de façon indépendante. On peut utiliser une troisième technique qui repose sur le calcul du temps de réponse. Le temps de réponse d'une tâche i inclut la durée J_i entre la demande du processeur et son obtention, la durée d'exécution C_i et la durée totale de sa préemption par d'autres tâches.

Dans [JP86, ABRT93], Joseph et Pandia, puis Audsley et al. ont proposé de résoudre l'équation suivante afin d'évaluer les temps de réponse d'un jeu de tâches périodiques :

$$r_i = J_i + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + r_i}{P_j} \right\rceil C_j \tag{3}$$

où $hp(i)$ est l'ensemble des tâches de plus haute priorité que i. L'équation (3) suppose que les délais critiques des tâches sont inférieures ou égales aux périodes. Avec l'équation (3), on peut vérifier qu'une tâche i respecte ses contraintes temporelles si $r_i \leq D_i$.

Lorsque les échéances sont arbitraires, le calcul du temps de réponse est plus complexe. Pour une tâche i, son activation courante peut être préemptée par ses activations précédentes et les tâches de plus forte priorité. En effet, pour chaque tâche, il faut tenir compte du fait qu'une activation n n'est pas forcément terminée alors que l'activation $n+1$ est prête à démarrer. En général, on fait l'hypothèse que l'activation $n+1$ d'une tâche ne peut commencer avant la terminaison de l'activation n.

Dans [Leh90], Lehoczky a proposé un calcul du temps de réponse qui tient compte de ce phénomène de rafale. Le temps de réponse est alors évalué de la façon suivante :

$$r_i = \max_{q=0,1,2,\ldots} (J_i + w_i(q) - qP_i) \tag{4}$$

avec

$$w_i(q) = (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{P_j} \right\rceil C_j$$

tel que :

$$\forall q : w_i(q) \geq (q+1).P_i$$

Dans cette partie, nous avons présenté les principaux tests de faisabilité applicables à des algorithmes préemptifs à priorités fixes en présence de tâches périodiques indépendantes. Dans la littérature, il existe bien d'autres travaux qui étendent les résultats précédemment décrits [Riv98, Leb98, SSNB95].

Toutefois, lorsque l'on souhaite étudier des systèmes utilisant un modèle de tâches différent ou des ordonnanceurs plus complexes, ces tests ne sont généralement plus applicables. En outre, il peut être long et fastidieux d'élaborer de nouveaux tests pour des modèles spécifiques de tâches ou d'ordonnanceurs. La présence de tampons pose un problème supplémentaire qui semble peu traité à ce jour. Dans la partie suivante, nous nous attachons à déterminer hors-ligne une borne sur leur taille, quel que soit l'ordonnanceur utilisé.

3 Dimensionnement des tampons

Nous supposons maintenant que des tâches périodiques partagent des tampons. Dans cet article, une tâche produit ou consomme un message à chacune de ses activations périodiques. Producteurs et consommateurs fonctionnent de façon asynchrone : ils sont réveillés à leur rythme. Un consommateur peut donc être activé sans pour autant disposer d'un message à traiter : l'activation courante du consommateur se termine alors immédiatement. Le consommateur ne reste pas bloqué en attente d'un message.

Le tampon fonctionne de façon FIFO : le premier message placé dans le tampon est aussi le premier à en être extrait.

L'application est faisable si les contraintes temporelles des tâches sont respectées et si aucun débordement de tampon n'intervient.

Concernant ce deuxième point, il existe des similitudes avec certains résultats issus du monde des réseaux hauts débits. C'est le cas des services de communication utilisés par le transport de la voix dans les réseaux ATM : la couche d'adaptation AAL2 [GK96].

Dans un premier temps, nous en décrivons le fonctionnement. Puis, nous appliquons la méthode de dimensionnement utilisée dans cet environnement à notre jeu de tâches périodiques.

3.1 A propos de la taille des tampons dans la couche AAL2 d'ATM

Nous regardons les systèmes constitués d'un processeur émetteur et d'un processeur récepteur connectés par un réseau ATM. L'émetteur transmet des données audio à une cadence fixe. Nous noterons ce débit d'émission d. Le débit est exprimé en cellule, l'unité de transfert dans un réseau ATM.

Le temps de transport de ces cellules de l'émetteur au récepteur est variable et est compris entre δ_{min} et δ_{max}, respectivement le délai minimum et maximum de transport. Cette variation du délai de transmission, ou *gigue*, est le fait du comportement temporel des différents noeuds (commutateurs ou brasseurs) traversés entre l'émetteur et le récepteur. Nous notons cette gigue $\triangle$ avec $\triangle = \delta_{max} - \delta_{min}$.

Se pose alors le problème de la livraison des données par le récepteur aux couches logicielles de plus haut niveau. En effet, il est nécessaire de restituer le flux de données audio à la cadence de l'émetteur.

Pour ce faire, on utilise un tampon dans lequel les informations audio sont mémorisées afin de supprimer la gigue produite par le réseau de transmission. Une des difficultés consiste alors à déterminer

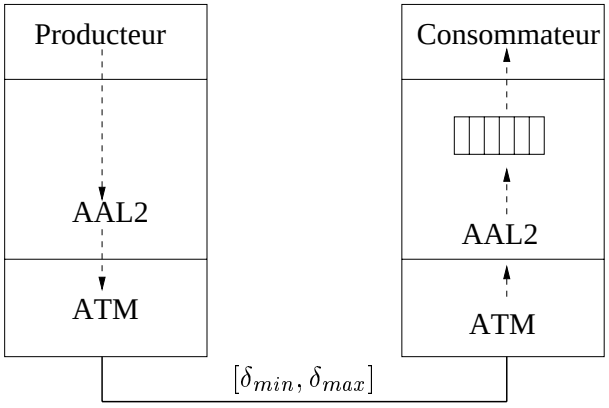

FIG. 2 – La couche AAL2

la taille de ce tampon afin d'éviter son débordement. Il a été montré que la taille maximum du tampon est de [GK96] :

$$B = \left\lceil \frac{2.\triangle}{d} \right\rceil \tag{5}$$

où $2.\triangle$ constitue **le plus grand délai pendant lequel les informations peuvent s'accumuler dans le tampon**. Dans ATM, ce délai est aussi le temps d'attente maximum d'une cellule dans le tampon (ou délai de mémorisation dans la suite de cet article) et aussi le plus grand délai sans consommation. Dans [GK96], il est démontré que cette borne est nécessaire pour éviter un débordement du tampon mais aussi suffisante car un tampon de plus grande taille est inutile.

3.2 De la couche AAL2 aux tâches périodiques

Nous revenons à notre système constitué de tâches périodiques. On étudie un ensemble de tâches qui partagent un tampon donné. On rappelle qu'une tâche produit ou consomme **un** message à chacune de ses activations. Pour simplifier l'explication de la méthode d'établissement des bornes, nous supposons ici qu'une tâche accède à un tampon et un seul. Nous définissons $PROD$, l'ensemble des tâches qui produisent dans un tampon et $CONS$, l'ensemble des tâches qui consomment des messages du tampon.

Un certain nombre de points communs existent entre les résultats présentés ci-dessus et les systèmes que nous étudions.

Les producteurs/consommateurs sont périodiques.

Contrairement à la couche AAL2, les producteurs et consommateurs interfèrent les uns par rapport aux autres de par l'accès concurrent au processeur.

Dans le cas d'ATM, la borne est déterminée grâce au plus grand délai d'accumulation qui correspond aussi au pire délai de mémorisation d'une cellule ou encore au plus grand délai sans consommation. Dans les systèmes que nous étudions, nous cherchons toujours le plus grand délai d'accumulation associé à un tampon. Toutefois, ce délai n'est plus équivalent au plus grand délai sans consommation. Rechercher une borne sur la taille d'un tampon consiste à déterminer la contribution de chaque producteur pendant le délai d'accumulation tout en tenant compte des consommations effectuées. Pour ce faire, nous déterminons une borne sur le délai de mémorisation des messages dans le tampon.

En outre, avant de pouvoir appliquer l'approche utilisée dans la couche AAL2, il existe une condition nécessaire supplémentaire dans notre cas, pour que la taille des tampons soit bornée. Nous la nommerons *"contrainte de débit"* . Cette condition implique que le débit en production soit inférieur ou égal au débit en consommation ; ou autrement dit, pour un tampon donné :

$$\sum_{prod \in PROD} \frac{1}{P_{prod}} \leq \sum_{cons \in CONS} \frac{1}{P_{cons}} \quad (6)$$

avec P_{cons} (resp. P_{prod}) les périodes des consommateurs (resp. producteurs).

Enfin, par la suite, nous supposons que toutes les tâches respectent leur échéance, c'est-à-dire que $\forall i : r_i \leq D_i$.

3.3 Délai maximum de mémorisation des messages dans un tampon

Pour appliquer la technique utilisée dans la couche AAL2, il est d'abord nécessaire de déterminer le délai maximum d'attente d'un message dans le tampon ou délai de mémorisation. Nous noterons δ ce délai.

Le délai de mémorisation est le délai entre la production d'un message et l'instant de sa consommation. Dans les systèmes considérés, les producteurs et les consommateurs interfèrent les uns par rapport aux autres de par l'accès concurrent au processeur.

Ainsi, selon la configuration du jeu de tâches et compte tenu du fait que le tampon fonctionne de façon FIFO, entre l'instant de production d'un message par un producteur i et l'instant de sa comsommation, y activations du consommateur sont nécessaires pour consommer les messages déjà présents dans le tampon. Nous ne cherchons pas, pour l'instant, à quantifier précisément y.

En outre, au pire cas, la production du message de i peut être réalisée immédiatement après l'activation du consommateur. De même, la consommation du message produit par i peut être effectuée en fin d'activation du consommateur. Le délai maximum de mémorisation est donc au pire cas de $P_{cons} + y.P_{cons} + D_{cons}$. Soit finalement, $\delta = (y+1).P_{cons} + D_{cons}$.

3.4 Dimensionnement des tampons : cas $\forall i : D_i \leq P_i$

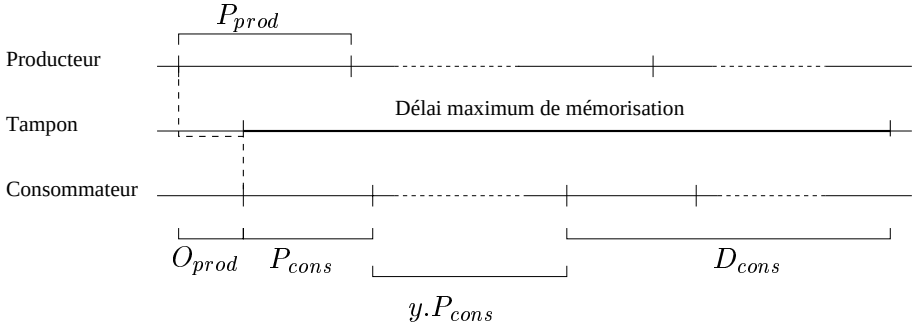

FIG. 3 – Désynchronisation des producteurs et consommateurs

A partir du délai de mémorisation maximum d'un message, on peut maintenant proposer des bornes sur les tampons.

Dans ATM/AAL2, le délai maximum de mémorisation démarre lors de l'envoi de la première cellule du flux. Or, dans notre cas, aucune hypothèse n'est faite sur l'instant où démarre le délai de mémorisation. Il existe donc une certaine désynchronisation entre les réveils des producteurs et des consommateurs. La borne maximale sur cette désynchronisation, notée O_{prod}, est le pire délai entre le réveil du consommateur, qui constitue le début du délai de mémorisation, et les réveils du producteur *prod* qui va émettre un message pendant ce délai de mémorisation (cf. figure 3). Comme durant cette désynchronisation, des messages peuvent être produits, avec un débit d'un message par période la taille maximum du tampon B sera de :

$$B = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{\delta + O_{prod}}{P_{prod}} \right\rceil - y \right) \tag{7}$$

La borne sur la taille du tampon est obtenue en prenant le maximum lorsque y varie. Notons que contrairement à la solution utilisée dans ATM, y messages sont retirés de la borne dans l'équation (7). En effet, la borne dans ATM est construite grâce au plus grand délai d'accumulation des messages dans le tampon qui est égal au plus grand délai sans consommation. Comme, dans notre cas, plusieurs producteurs interfèrent entre eux, ces deux délais ne sont plus équivalents. Pendant le délai de mémorisation, y messages sont consommés. Ils doivent donc être retirés afin d'obtenir le nombre de message accumulé pendant le plus grand délai d'accumulation,

Nous allons successivement étudier les cas de figure où un tampon est partagé par un producteur et un consommateur, puis par N producteurs et un consommateur et enfin par N producteurs et M consommateurs. Dans tous ces cas, nous supposons que $D_{cons} = P_{cons}$. Le délai maximum de mémorisation d'un message devient $\delta = (y+2).P_{cons}$.

Au pire cas, lorsque le tampon est partagé par un producteur et un consommateur uniquement, la contrainte de débit (6) implique que $P_{cons} = P_{prod}$. En outre, comme producteur et consommateur ont une période identique, nous avons $O_{prod} = 0$. Si l'on substitue ces informations à (7), nous obtenons $B = \left\lceil \frac{(y+2).P_{cons}}{P_{cons}} \right\rceil - y = y + 2 - y$. Et donc :

$$B = 2 \tag{8}$$

Pour N producteurs et un consommateur, les réveils sont désynchronisés. Au pire cas, la durée de cette désynchronisation est $\forall prod \in PROD : O_{prod} = P_{prod}$. Nous avons toujours $\delta = (y+2).P_{cons}$. En substituant à (7), nous obtenons $B = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons}+P_{prod}}{P_{prod}} \right\rceil - y \right)$. Soit :

$$B = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons}}{P_{prod}} \right\rceil + N - y \right) \tag{9}$$

Enfin, avec M consommateurs et N producteurs, le pire délai maximal de mémorisation est déterminé à partir de la plus grande période des consommateurs. Notons $P_{cons}^{\max}$ la plus grande période des consommateurs. Nous avons, $\delta = (y+2).P_{cons}^{\max}$. Comme, $\forall prod \in PROD : O_{prod} = P_{prod}$, nous obtenons :

$$B = \max_{\forall y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{(y+2).P_{cons}^{\max}}{P_{prod}} \right\rceil + N - y \right) \tag{10}$$

3.5 Généralisation lorsque D_i est quelconque vis-à-vis de P_i

Lorsque les délais critiques sont plus grands que les périodes, selon l'ordonnancement des tâches, les différentes activations d'un producteur peuvent être retardées et provoquer une rafale de production de messages (cf. figure 4).

Précédemment, la désynchronisation maximale était bornée par la période des producteurs. Cette désynchronisation maximale représentait la production au plus tard de message pendant une activation et une seule. Avec un délai critique éventuellement supérieur à la période, au pire cas, cette désynchronisation est bornée par $\forall prod \in PROD : O_{prod} = D_{prod}$.

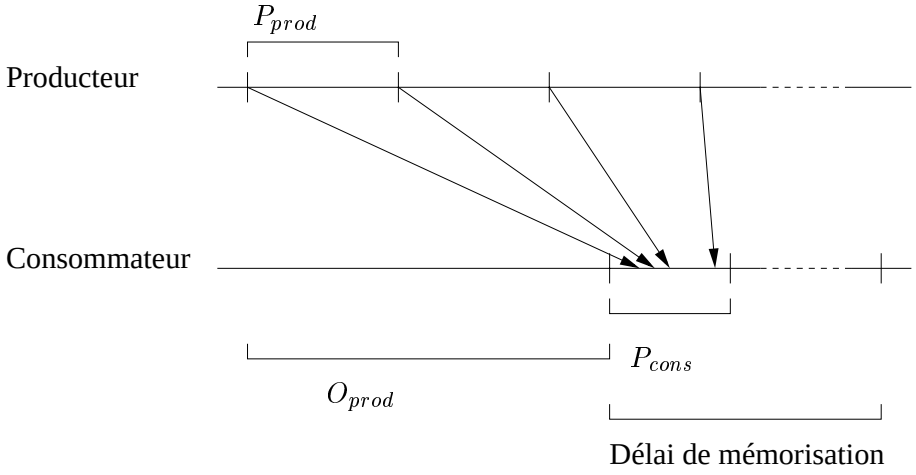

FIG. 4 – Rafale de messages

Le plus grand délai de mémorisation devient $\delta = \forall cons \in CONS : \max((y+1).P_{cons} + D_{cons})$. Finalement la borne sur la taille du tampon est de :

$$B = \max_{y \geq 0} \left(\sum_{prod \in PROD} \left\lceil \frac{\forall cons \in CONS : \max((y+1).P_{cons} + D_{cons}) + D_{prod}}{P_{prod}} \right\rceil - y \right) \tag{11}$$

3.6 Propriétés de la proposition (9)

Nous regardons plus particulièrement le cas de la proposition (9). Le système considéré comprend un consommateur et N producteurs. Ce paradigme est classique dans les systèmes puisqu'il correspond, par exemple, au multiplexage des requêtes de plusieurs clients (les producteurs) vers un serveur (le consommateur). Il se trouve que dans les applications que nous ciblons, ce paradigme est aussi fréquemment présent. Dans ce paragraphe, nous discutons d'une propriété remarquable de la proposition (9), qui est la suivante :

Propriété 1 (Bornes spécifiques) *Dans le cas $\forall i : D_i \leq P_i$, lorsque $\forall i : r_i \leq D_i$, pour un tampon avec un consommateur et N producteurs, la borne est de :*

$$B = 2.N \tag{12}$$

si les tâches sont harmoniques [1]*. La borne sur la taille du tampon est de*

$$B = 2.N + 1 \tag{13}$$

dans le cas contraire.

On commence par étudier le cas où les tâches ne sont pas harmoniques. On montre au pire cas, grâce aux équations (6) et (7), qu'un seul producteur peut accumuler 3 messages et que les $N-1$ producteurs restant accumulent 2 messages.

Nous montrons d'abord qu'un producteur peut produire $y+3$ messages au plus pendant le délai de mémorisation. Pour un jeu de tâches composé de 1 consommateur et N producteurs, nous savons

[1] Un ensemble de tâches est harmonique si toutes les tâches possèdent des périodes multiples entre elles.

que $P_{prod} > P_{cons}$. Le débit maximum d'un producteur i est obtenu lorsque $P_i \to P_{cons}$. Soit une production de $\left\lceil \frac{(y+2).P_{cons}+O_i}{P_i} \right\rceil = \left\lceil \frac{(y+2).P_{cons}+P_{cons}}{P_{cons}} \right\rceil = y+3$ messages maximum pendant δ.

Nous cherchons maintenant à déterminer l'intervalle I des valeurs de P_i pour lequel la production reste de $y+3$ messages. Lorsque $P_i \to P_{cons}$, le premier (ou le dernier) des $y+3$ messages est produit dans l'intervalle $]0, P_{cons}[$. Ce délai peut être uniformément réparti aux $y+1$ instances du producteur afin de maximiser sa période tout en conservant une production de $y+3$ messages. Ainsi, la plus grande période du producteur i, tel que $y+3$ messages sont produits, est donc $P_i = \frac{P_{cons}}{y+1} + P_{cons} = \frac{(y+2).P_{cons}}{y+1}$. La production est de $y+3$ messages pour un producteur lorsque sa période est comprise dans l'intervalle $I =]P_{cons}, \frac{(y+2).P_{cons}}{y+1}[$.

Nous montrons maintenant que si la période d'un producteur tend vers $\frac{(y+2).P_{cons}}{y+1}$, alors les $N-1$ producteurs restant ont une période qui tend vers $(y+2).P^+_{cons}$. En effet, la contrainte de débit (6) peut être exprimée de la façon suivante :

$$\frac{1}{x} + \sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq \frac{1}{P_{cons}}$$

ou encore :

$$\sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq \frac{x - P_{cons}}{x.P_{cons}}$$

avec $PROD^*$, l'ensemble des producteurs $PROD$ auquel nous avons retiré le producteur dont la période est x. Ce producteur est celui qui produit $y+3$ messages.

Posons :

$$f(x) = \frac{x - P_{cons}}{x.P_{cons}}$$

et étudions les limites de cette fonction quand $x \to \frac{(y+2).P_{cons}}{y+1}$ et $x \to P_{cons}$. On obtient :

- $\lim_{x \to \frac{(y+2).P_{cons}}{y+1}} f(x) = \frac{1}{(y+2).P_{cons}}$
  Or $\sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq \frac{1}{(y+2).P_{cons}}$, ce qui implique, au pire cas : $\forall prod \in PROD^* : P_{prod} \to (y+2).P^+_{cons}$.
- $\lim_{x \to P_{cons}} f(x) = 0$
  Or $\sum_{prod \in PROD^*} \frac{1}{P_{prod}} \leq 0$, ce qui implique, au pire cas : $\forall prod \in PROD^* : P_{prod} \to \infty$.

L'intervalle des périodes des producteurs appartenant à $PROD^*$ est donc $J =](y+2).P_{cons}, \infty[$. Nous obtenons donc une production maximum de :

$$\sum_{prod \in PROD^*} \left\lceil \frac{\delta + O_{prod}}{P_{prod}} \right\rceil$$

ou encore :

$$\sum_{prod \in PROD^*} \left\lceil \frac{(y+2)P_{cons} + (y+2).P^+_{cons}}{(y+2).P^+_{cons}} \right\rceil = 2.N - 2$$

Finalement, d'après (7), la borne est de :

$$B = y + 3 + \sum_{prod \in PROD^*} \left\lceil \frac{(y+2)P_{cons} + (y+2).P^+_{cons}}{(y+2).P^+_{cons}} \right\rceil - y$$

et donc $B = y + 3 + 2.N - 2 - y = 2.N + 1$

Dans le cas d'un jeu de tâches harmoniques, nous appliquons la méthode décrite ci-dessus. Cette fois-ci, un producteur émet au pire cas $y+2$ messages pendant le délai de mémorisation. En effet, dans le cas harmonique, le décalage entre les activations de 2 tâches ayant la même période est nulle. Un producteur i émet donc au pire cas $\left\lceil \frac{(y+2).P_{cons}}{P_{cons}} \right\rceil = y+2$ messages. L'intervalle I des valeurs possibles de P_i devient $[P_{cons}, \frac{(y+2).P_{cons}}{y+1}]$. Si l'on substitue à (7) ces informations comme pour le cas non harmonique, on obtient :

$$B = y + 2 + \sum_{prod \in PROD^*} \left\lceil \frac{(y+2)P_{cons} + (y+2).P_{cons}}{(y+2).P_{cons}} \right\rceil - y$$

et donc $B = y + 2 + 2.N - 2 - y = 2.N$

3.7 Conclusions

A partir du fonctionnement de la couche AAL2, nous avons proposé un certain nombre de bornes pour dimensionner des tampons partagés par un ensemble de tâches périodiques.

Ces bornes ne supposent pas l'utilisation d'un algorithme d'ordonnancement particulier. Elles peuvent donc être affinées si des hypothèses sont faites sur l'ordre d'exécution des différentes tâches.

4 Outils et étude de cas

Nous illustrons maintenant les propositions de la partie 3 grâce à une étude de cas. Nous commençons par la description d'un outil de simulation que nous avons réalisé en guise de démonstrateur. Cet outil implante les méthodes décrites dans les parties 2 et 3 de cet article. Ensuite, nous étudions une application fictive. Nous vérifions le respect des contraintes temporelles de ses tâches et surtout, nous établissons les bornes sur les tailles des tampons.

4.1 L'outil de simulation Cheddar

Cheddar est un outil de simulation dont les objectifs sont doubles :

1. Offrir un outil éducatif abordant les principaux résultats traités dans un cours d'initiation à l'ordonnancement temps réel.
2. Offrir une plate-forme permettant de prototyper rapidement de nouveaux algorithmes d'ordonnancement ou de nouvelles méthodes d'évaluation de la faisabilité.

Pour ce faire, l'outil est constitué de deux composants relativement indépendants :

- Un éditeur graphique permettant de décrire un système temps réel.
- Une bibliothèque orientée objets contenant les principaux algorithmes d'ordonnancement et d'analyse de la faisabilité proposés dans la littérature. Cette bibliothèque pouvant être utilisée de façon indépendante vis-à-vis de l'éditeur graphique, pour la mise en oeuvre de simulations ad-hoc par exemple.

Cheddar est diffusé sous la licence GNU/GPL. A ce titre, les sources et les exécutables sont disponibles à partir du site Web http://beru.univ-brest.fr/~singhoff/cheddar. Pour des raisons de portabilité, les deux composants sont réalisés en Ada/Gtk. Actuellement, Cheddar fonctionne sous Windows, Linux et Solaris.

L'outil permet de décrire un système en termes de tâches périodiques ou non, de processeurs, de messages, de ressources partagées et de tampons.

A partir de cette description, l'outil analyse la faisabilité du système et peut simuler l'ordonnancement du jeu de tâches. Différentes informations sont délivrées à l'utilisateur afin de lui permettre d'affiner sa compréhension du comportement de son système (taux d'utilisation, période d'étude, temps de

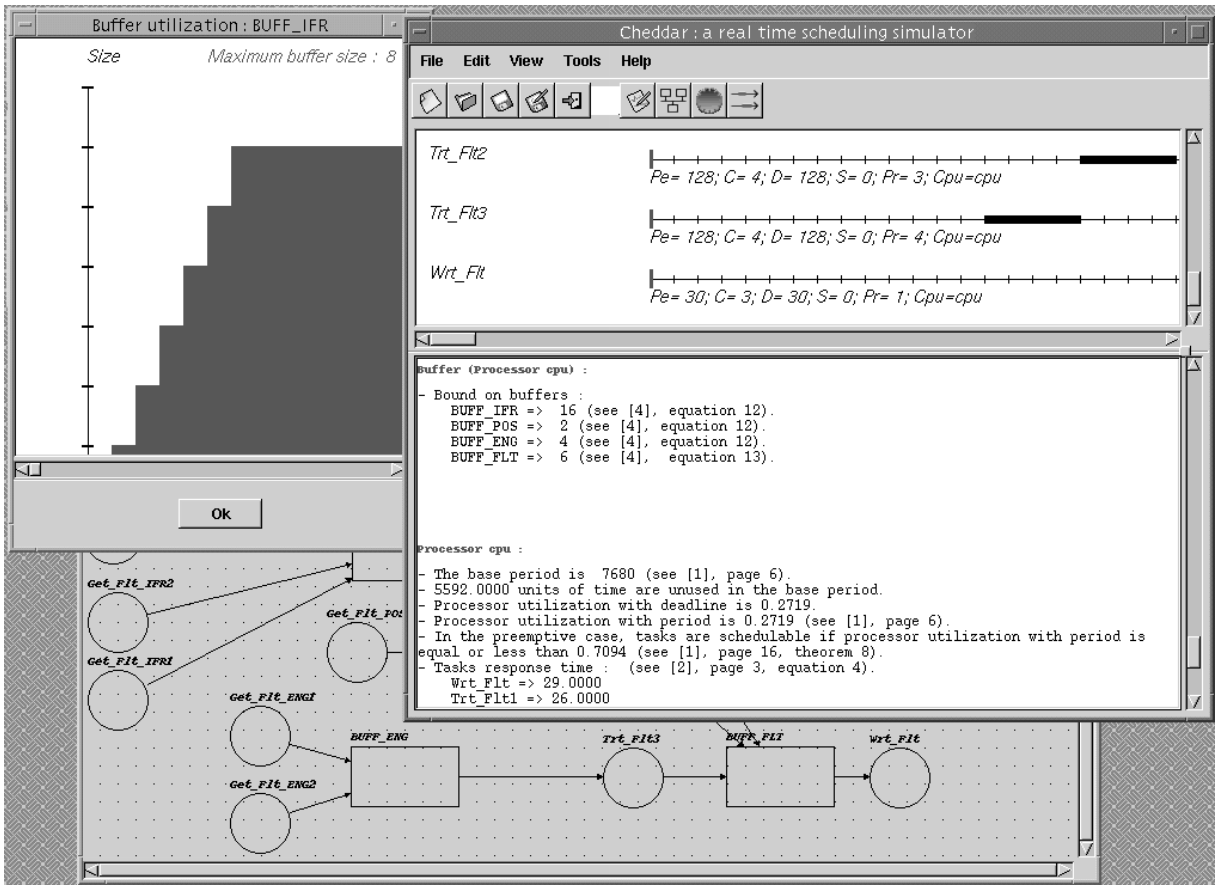

FIG. 5 – L'outil de simulation Cheddar

blocage sur les ressources partagées, période non occupée du processeur, etc). Comme dans [KRP+94], chaque résultat est accompagné de la référence bibliographique décrivant la méthode de calcul utilisée.

Les principaux algorithmes d'ordonnancement disponibles à ce jour sont Rate Monotonic, Deadline Monotonic, Earliest Deadline First, Least Laxity First. Les différentes politiques d'ordonnancement normalisées par POSIX.4 sont implantées (SCHED_RR, SCHED_FIFO et SCHED_OTHERS [2]).

Les outils d'analyse de la faisabilité sont basés sur les méthodes décrites dans la partie 2 de cet article. Pour les ressources partagées, les protocoles disponibles sont PCP et PIP [SRL90].

Enfin, si des tampons sont définis par l'utilisateur, Cheddar calcule leur borne conformément aux propositions de la partie 3 et permet, lors d'une simulation, d'observer l'évolution de leur occupation.

4.1.1 Contraintes de précédences entre les tâches

L'outil offre la possibilité de spécifier des contraintes de précédence entre les tâches. Ces contraintes peuvent être exploitées durant la simulation de l'ordonnancement et dans les tests de faisabilité. Elles sont de deux types différents : les contraintes liant deux tâches par l'échange d'un message ou plus simplement, par l'existence d'une contrainte sur l'ordre d'exécution des deux tâches.

Le simulateur offre plusieurs méthodes permettant de manipuler ces contraintes de précédence. En particulier, il est possible d'appliquer les méthodes proposées par Chetto et Blazewicz [Bla76, CSB90]. Ces techniques consistent à modifier priorités et délais critiques des tâches afin de garantir un ordonnancement respectant les contraintes de précédence.

[2] Dans Cheddar, SCHED_OTHERS est un ordonnanceur à temps partagé.

$\forall i : J_i := 0, r_i := 0, r'_i := 0;$
$\forall i :$ Calculer_temps_de_réponse (r_i) ;
Tant que $(\exists i : r_i \neq r'_i)$ {
  $\forall i : J_i := max(J_i, \forall j \; avec \; j \prec i : r_j + M_{ji});$
  $\forall i : r'_i := r_i;$
  $\forall i :$ Calculer_temps_de_réponse (r_i) ;
}

FIG. 6 – Calcul Holistique

Avec les contraintes de précédence, les tâches peuvent être organisées en chaîne de traitements. Une chaîne de traitements est constituée de plusieurs tâches exécutées successivement. Ces tâches sont éventuellement placées sur des processeurs différents et communiquent par échange de messages. Ici, il ne s'agit plus de vérifier le respect d'une contrainte qui est locale à un processeur donné, mais de vérifier un délai de bout en bout : on cherchera à contrôler, par exemple, que le délai entre le démarrage de la première tâche et la terminaison de la dernière tâche d'une chaîne de traitements est inférieur à un délai critique donné.

Le calcul du pire délai de bout en bout est réalisé par un calcul de proche en proche des temps de réponse des tâches : c'est le calcul holistique proposé initialement par Tindell [TC94].

La figure 6 décrit la méthode de calcul des délais de bout en bout utilisée dans l'outil. Elle consiste à considérer le temps de réponse d'une tâche i comme la gigue sur activation d'une tâche j (paramètre J_j) lorsque la tâche j doit être déclenchée après terminaison de i. Quand une tâche j possède plusieurs prédécesseurs dans le graphe de précédence, le plus grand temps de réponse des prédécesseurs est considéré. Modification des gigues et calculs des temps de réponse sont réalisés jusqu'à convergence de ces derniers. Dans la figure 6, M_{ji} désigne le pire délai d'acheminement d'un message. L'opérateur $\prec$ exprime l'existence d'une contrainte de précédence entre deux tâches successives dans une chaîne de traitements.

4.1.2 Etude de la faisabilité par simulation

De nombreux tests de faisabilité ont été proposés dans la littérature. Toutefois, ces tests sont dédiés à un nombre limité de modèles de tâches et d'ordonnanceurs. Lorsque l'on souhaite étudier des systèmes utilisant un modèle de tâches différent ou des ordonnanceurs plus complexes, ces tests ne sont généralement plus applicables.

En outre, il peut être long et fastidieux d'élaborer de nouveaux tests pour des modèles spécifiques de tâches ou d'ordonnanceur. Cheddar propose quelques outils afin d'étudier le comportement de tels systèmes.

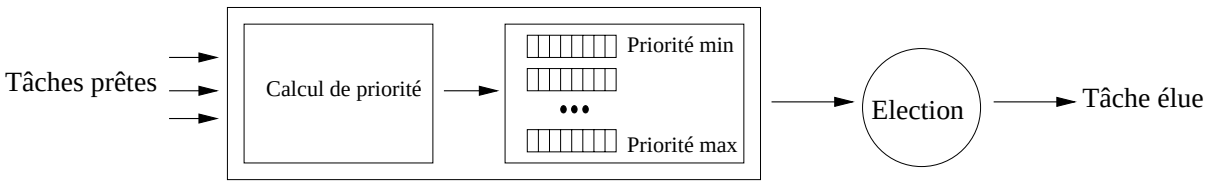

FIG. 7 – Fonctionnement des ordonnanceurs dans Cheddar

Ces outils sont construits à l'aide d'un simulateur dont l'algorithme d'ordonnancement est paramétrable. Une fois l'ordonnanceur paramétré, il est possible d'effectuer des simulations dont les résultats peuvent être analysés, pour extraire, par exemple, une estimation des temps de réponse des tâches. **La validité de ces temps de réponse est bien sûr limitée à la simulation effectuée.**

```
1   declare :
2       dynamic_priority : integer_task_array ;
3
4   priority_section :
5       dynamic_priority := start_time
6           + ((activation_number-1)*period)
7           + deadline ;
8
9   election_section :
10      return min(dynamic_priority) ;
```

FIG. 8 – Ordonnanceur défini par un utilisateur

Dans Cheddar, tous les ordonnanceurs sont découpés en trois traitements distincts (cf. figure 7). Le premier traitement consiste à calculer une priorité pour chaque tâche prête. Le deuxième traitement maintient une file d'attente par niveau de priorité gérée selon les politiques POSIX.4 *SCHED_FIFO* et *SCHED_RR*. Enfin, le troisième traitement effectue l'élection de la tâche parmi toutes les tâches positionnées en tête des files d'attente.

Afin de pouvoir expérimenter de nouveaux algorithmes d'ordonnancement ou d'étudier des systèmes comportant des modèles de tâches spécifiques, il est possible de redéfinir, grâce à un langage simple, les étapes de calcul de la priorité et d'élection.

La figure 8 montre succinctement la façon dont ces étapes peuvent être redéfinies par l'utilisateur. Dans cet exemple, il s'agit de l'algorithme EDF dont le principe consiste à calculer pour chaque tâche une échéance, puis, d'élire la tâche dont l'échéance est la plus proche.

Le calcul de l'échéance de chaque tâche est effectué grâce aux lignes 4 à 7. Pour décrire ce calcul, l'utilisateur dispose des paramètres de tâches, d'informations collectées par le simulateur [3] ou encore de variables définies au sein de l'ordonnanceur (cf. lignes 1 et 2 de la figure 8). En outre, il est possible, grâce à l'éditeur graphique, d'affecter aux tâches du système des paramètres définis par l'utilisateur et exploitable dans l'ordonnanceur paramétrable.

L'élection de la tâche à exécuter est réalisée entre les lignes 9 et 10. On élit ici la tâche dont l'échéance est la plus petite.

Les lignes 4 à 10 sont interprétées par le simulateur à chaque fois que les tâches doivent être réordonnancées ; c'est-à-dire à chaque unité de temps pour les algorithmes préemptifs et à chaque fois qu'une tâche libère le processeur dans le cas non-préemptif.

Nous renvoyons le lecteur à [Tea03] pour une description plus exhaustive de ce service ainsi que des fonctionnalités offertes par Cheddar.

4.2 Etude de cas

Afin d'illustrer les parties 2 et 3 de cet article, nous allons étudier une application temps réel fictive de diagnostic de pannes. On suppose qu'une application de diagnostic est implémentée sur un robot mobile. Ce robot est constitué de 2 moteurs, de 8 capteurs de proximité infra-rouge et d'un détecteur de position. L'objectif de l'application est de stocker en mémoire les informations relatives aux pannes détectées sur ces composants. Le compte rendu des pannes est ensuite utilisé par l'opérateur afin de déterminer les élements défectueux à réparer ou à remplacer.

On suppose que notre application est constituée d'un ensemble de tâches ordonnancées par un algorithme à priorités fixes. On suppose, en outre, que ces tâches partagent des informations à l'aide de tampons.

[3] temps processeur consommé, numéro d'activation pour les tâches périodiques, ressources partagées utilisées, ...

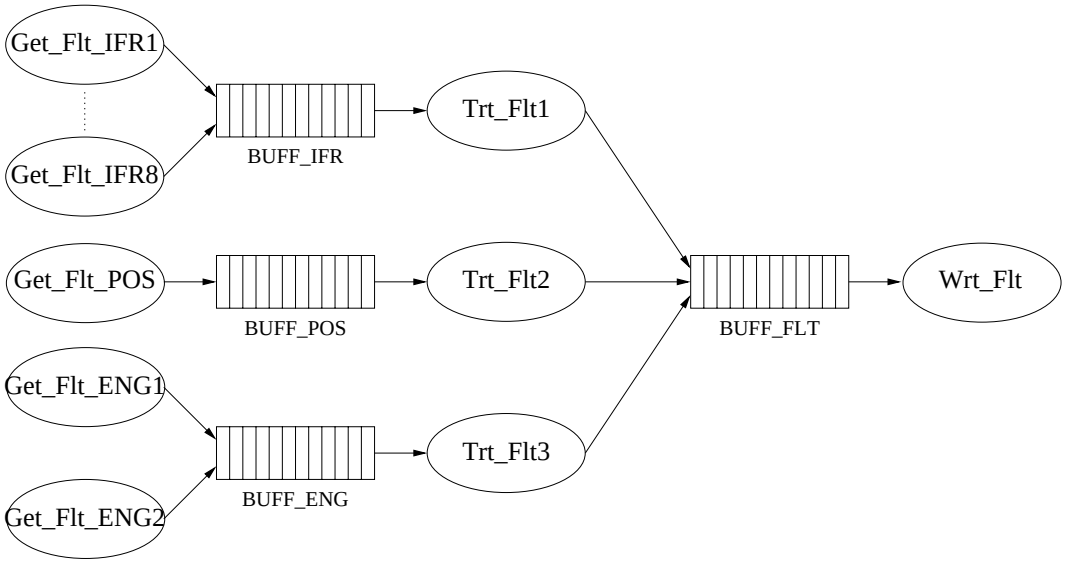

FIG. 9 – Tâches, tampons et flot des données

Tâches et tampons sont organisés en flot de données. Les sources de ce flot constituent les points d'entrée des informations captées par le système. Le puits de ce flot modélise la tâche qui finalement stocke en mémoire stable le diagnostic (cf. figure 9).

Les tâches Get_Flt_IFRx ($x \in [1..8]$), Get_Flt_ENGx ($x \in [1..2]$) et Get_Flt_POS récupèrent de manière périodique respectivement, l'état des capteurs infrarouges, l'état des moteurs et l'état du capteur de position. Ces informations, transmises par l'intermédiaire des tampons BUFF_IFR, BUFF_POS et BUFF_ENG, sont analysées par les tâches Trt_Flt$x$ qui détectent les pannes éventuelles et les formatent pour un affichage. Finalement, ces données formatées sont placées dans le tampon BUFF_FLT afin d'être écrites en mémoire stable par la tâche Wrt_Flt.

Nom	P_i/D_i	C_i	Priorité	r_i
Get_Flt_ENG1	256	2	6	12
Get_Flt_ENG2	256	2	7	10
Get_Flt_IFR1	512	1	8	8
Get_Flt_IFR2	512	1	9	7
Get_Flt_IFR3	512	1	10	6
Get_Flt_IFR4	512	1	11	5
Get_Flt_IFR5	512	1	12	4
Get_Flt_IFR6	512	1	13	3
Get_Flt_IFR7	512	1	14	2
Get_Flt_IFR8	512	1	15	1
Get_Flt_POS	128	2	5	14
Trt_Flt1	64	4	2	26
Trt_Flt2	128	4	3	22
Trt_Flt3	128	4	4	18
Wrt_Flt	30	3	1	29

FIG. 10 – Paramètres des tâches

Concernant les paramètres des tâches, nous formulons les hypothèses suivantes :
– Les périodes des tâches Trt_Flt$x$ et Wrt_Flt sont déduites à partir des périodes des tâches

d'acquisition conformément à la contrainte de débit (6).
- Nous considérons le cas où les tâches qui produisent les informations sont plus prioritaires que les tâches qui consomment ces informations.

Les paramètres des tâches sont résumés dans le tableau 10. Dans cet article, le niveau de priorité 1 constitue le niveau de priorité le plus faible.

Nom	Equation utilisée	Borne
BUF_IFR	(12)	16
BUF_POS	(12)	2
BUF_ENG	(12)	4
BUF_FLT	(13)	7

FIG. 11 – Bornes sur les tampons

Nous nous intéressons maintenant à la faisabilité de ce système. Pour ce faire, nous pouvons :

1. Dans un premier temps, vérifier que les contraintes temporelles des tâches sont respectées ; ou en d'autres termes que $\forall i : r_i \leq D_i$. L'outil vérifie ce premier point grâce à l'équation (4). Si des échéances ne sont pas respectées, elles sont signalées. Les temps de réponse ainsi calculés sont donnés dans le tableau 10.
2. Dans un deuxième temps, déterminer les bornes sur les différents tampons de l'application. L'outil utilise les équations (12) et (13) pour effectuer cette deuxième opération.

L'application est valide si les temps de réponse sont inférieurs aux délais critiques et si les tailles de tampon spécifiées sont inférieures aux bornes calculées.

D'autre part, par simulation, l'utilisateur peut compléter son analyse par des histogrammes qui indiquent le taux d'occupation des tampons ainsi que des chronogrammes de l'ordonnancement calculé.

5 Conclusions

Dans cet article, nous avons présenté les objectifs du projet IRMA. Dans le projet IRMA, nous nous intéressons aux applications de supervision présentes dans les systèmes temps réel de grande taille. Par application de supervision, nous entendons les applications qui collectent les pannes et anomalies du système supervisé pour les analyser et finalement les présenter à l'opérateur lors des phases de maintenance du système.

En première approche nous avons supposé que les événements de panne étaient délivrés de façon périodique à l'application de supervision. D'une part, nous avons présenté les méthodes classiques de test de faisabilité permettant de vérifier a priori le respect des contraintes temporelles des tâches. D'autre part, nous avons étudié les bornes sur la taille des tampons dans le cas où toutes les tâches sont activées de façon périodique. Les bornes proposées sont valides quel que soit l'ordonnanceur utilisé. Elles permettent d'étudier de façon indépendante la faisabilité des tâches et le dimensionnement des tampons.

La suite du projet IRMA comprend deux volets. Dans un premier temps, nous souhaitons étudier de manière plus approfondie les bornes sur la taille des tampons partagés par des tâches périodiques. Notamment lorsque les délais critiques sont arbitraires et que plusieurs consommateurs utilisent un même tampon.

Dans un deuxième temps, nous prévoyons de lever l'hypothèse de périodicité sur la livraison des informations de pannes. En effet, la fiabilité des équipements est généralement exprimée selon des lois dont la nature est aléatoire [KS97]. Les problèmes soulevés ici sont nombreux mais deux d'entre eux nous semblent importants.

Le premier réside dans le dimensionnement des tampons. Peu de travaux ont étudié ce point lorsque les producteurs/consommateurs d'un tampon peuvent être déterministes/périodiques et aléatoires.

Un deuxième problème classique est celui du partage du processeur afin de garantir les contraintes temporelles des tâches périodiques tout en qualifiant la qualité de service offerte aux autres tâches. Ce dernier point a déjà été largement étudié, dans le cadre d'applications multimédias par exemple [SD02], mais peu de travaux supposent que les tâches des deux familles partagent des ressources (tampons et autres).

6 Remerciements

Nous remercions le Conseil Régional de Bretagne qui a participé au financement du projet IRMA. Merci à Nicolas Rivierre pour sa contribution sur la partie 4.1.2.

Références

[ABRT93] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 284–292, 1993.

[Bla76] J. Blazewicz. Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines. In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems, Amsterdam, Noth-Holland, 1976.

[Bur00] B. Burchell. A3XX Maintenance : A First Look. *Overhaul and Maintenance Revue. URL : www.aviationnow.com*, August 2000.

[CDKM00] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Ordonnancement temps réel.* Hermès, 2000.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints . *Real Time Systems, The International Journal of Time-Critical Computing Systems*, 2(3) :181–194, September 1990.

[Div] Vermont Air Pollution Control Division. On Board Diagnostics (OBD). Technical report, APCD Brochures.

[GK96] M. Gagnaire and D. Kofman. *Réseaux Haut Débit : réseaux ATM, réseaux locaux, réseaux tout-optiques.* Masson-Inter Editions, Collection IIA, 1996.

[Inc98] Sogitec Industries Inc. Gestion de maintenance intégrée pour les Falcon. *Revue Interactions*, (17) :6–9, September 1998.

[JP86] M. Joseph and P. Pandya. Finding Response Time in a Real-Time System. *Computer Journal*, 29(5) :390–395, 1986.

[KRP+94] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real Time Analysis.* Kluwer Academic Publishers, 1994.

[KS97] C. M. Krishna and K. G. Shin. *Real-Time Systems.* Mc Graw-Hill International Editions, 1997.

[Leb98] L. Leboucher. *Algorithmique et Modélisation pour la Qualité de Service des Systèmes Répartis Temps Réel.* Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications de Paris, septembre 1998.

[Leh90] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. pages 201–209. in Proc. 11th IEEE Real Time Systems Symposium, Lake Buena Vista, December 1990.

[LL73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1) :46–61, January 1973.

[LM80] J.Y.T Leung and M.L. Merril. A note on preemptive scheduling of periodic real time tasks. *Information processing Letters*, 3(11) :115–118, 1980.

[Riv98] N. Rivierre. *Ordonnancement temps réel centralisé, les cas préemptifs et non préemptifs*. Thèse de doctorat, Université de Versailles Saint Quentin, février 1998.

[SD02] F. Singhoff and I. Demeure. Synthèse sur la gestion des ressources dans les systèmes multimédias répartis. Rapport technique numéro singhoff-03-02, LIMI/EA 2215, Université de Brest, mars 2002.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols : An Approach to real-time Synchronization. *IEEE Transactions on computers*, 39(9) :1175–1185, 1990.

[SSNB95] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. Implications of Classical Scheduling Results For Real-Time Systems. *IEEE Computer*, 28(6) :16–25, June 1995.

[TC94] K. W. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3) :117–134, April 1994.

[Tea03] LIMI/EA 2215 Team. Cheddar online User's Guide. Technical report, Available from http ://beru.univ-brest.fr/~singhoff/cheddar/ug.html, March 2003.