

Collaboration entre méthode d'ordonnancement et calcul réseau

Marc Boyer, David Doose

ONERA

Résumé Dans cet article, nous faisons collaborer deux méthodes de calcul du monde temps réel : l'ordonnancement de tâches et le calcul réseau. L'ordonnancement de tâches cherche à savoir si, sur un processeur partagé, un ensemble de tâches (temps-réel) respectera ses échéances. Les différentes techniques d'ordonnancement peuvent souvent trouver exactement le pire temps de réponse, mais avec une complexité algorithmique assez importante. Le calcul réseau, lui, fait souvent des sur-approximations, mais avec une complexité algorithmique plus faible (souvent linéaire) qui lui permet de traiter des systèmes de très grande taille. Nous nous proposons ici de faire collaborer ces deux techniques dans un système embarqué communiquant en appliquant l'ordonnancement dans les calculateurs pour évaluer au plus près le flots de données qui en est issu dans un format adapté au calcul réseau.

1 Introduction

Les systèmes embarqués temps réels sont de nos jours constitués de dizaines voire centaines de calculateurs, hébergeant chacun des dizaines ou centaines d'applications temps réels. Pour garantir la correction de ces applications, il faut non seulement borner le temps de réponse des systèmes, mais aussi les délais subis par les données échangées à travers le réseau.

Pour calculer des temps de réponse, les techniques d'ordonnancement sont appliquées sur les calculateurs. Elles permettent souvent de calculer l'exact pire temps de réponse, au prix d'une forte complexité algorithmique. Cela n'est pas trop gênant pour les systèmes embarqués, où le nombre d'applications par calculateur reste raisonnable (un système trop complexe à analyser serait trop complexe à ordonnancer par le système).

En ce qui concerne le réseau, ce sont des dizaines de milliers de flux qu'il faut prendre en compte dans un avion comme l'A380, sur une topologie contenant une dizaine de commutateurs et une centaine de calculateurs. Pour borner le temps de traversée d'un tel réseau, il faut des méthodes avec une complexité algorithmique plus faible. Le calcul réseau [8] est capable d'utiliser des algorithmes linéaires, au prix d'approximation pessimistes. Il a été utilisé pour garantir les délais du réseau de l'A380 [6].

Pour garantir des délais sur un réseau partagé, le calcul réseau a besoin de contrats sur les trafics entrant (dits *courbes d'arrivée*) et aussi sur les capacités des commutateurs (dits *courbes de service*).

Notre idée est d'utiliser les informations d'ordonnement pour calculer au plus juste le trafic issu de chaque ordinateur. Grossièrement, lorsque le calcul réseau estime le trafic émis par une tâche périodique, il considère la somme des tailles maximales de trames émises par période, et en déduit un débit crête (une rafale) égal à la somme des tailles de trames, et un débit moyen qui est la rafale divisée par la période. Pour prendre en compte le trafic émis par plusieurs applications hébergées sur le même ordinateur, il fait la somme des débits individuels. Mais ce faisant, il considère que toutes les applications peuvent émettre leur rafale en même temps, ce qui est impossible, une seule application s'exécutant à un moment donné sur un processeur. Notre contribution consiste à considérer uniquement les ordonnancements possibles, d'en déduire les trafics générés, puis d'en déduire un contrat de trafic le plus juste possible.

Nous commencerons par faire une rapide présentation du calcul réseau, au paragraphe 2, à la fin de laquelle nous présenterons l'intuition qui fonde notre approche (paragraphe 2.3). Nous présenterons ensuite notre méthode d'ordonnement pour le calcul de courbes de trafic (paragraphe 3). Des exemples permettant d'illustrer le gain de la méthode sont présentés au paragraphe 4.

2 Le calcul réseau

L'objectif du calcul réseau est de permettre de calculer des bornes supérieures garanties pour les délais subis par des flux dans des réseaux, ainsi que pour les quantités de mémoire utilisées par ces flux dans les éléments de réseau. Pour ce faire, il modélise un contrat de trafic par des « courbes d'arrivée », et modélise le serveur par une « courbe de service » [4,5,8,3]. Nous n'entrerons pas dans les détails de cette théorie, mais uniquement les parties nécessaires pour cet article. On pourra se reporter à [2] pour une introduction en français.

2.1 Algèbre (min,plus)

Un des intérêts du calcul réseau est que les notions qu'il manipule s'expriment très bien dans le dioïde $(\min, +)$, dont la première loi est le minimum, et la seconde loi la somme.

Les flux de données sont représentés par leur fonction de cumul. Nous manipulons donc des fonctions croissantes (\mathcal{G}), nulles sur les négatifs (\mathcal{F}) et nulles en 0 (\mathcal{F}_0).

$$\mathcal{G} \stackrel{\text{def}}{=} \{f : \mathbb{R} \rightarrow \mathbb{R} \cup \{+\infty\} \mid x < y \implies f(x) \leq f(y)\}$$

$$\mathcal{F} \stackrel{\text{def}}{=} \{f \in \mathcal{G} \mid x < 0 \implies f(x) = 0\} \quad \mathcal{F}_0 \stackrel{\text{def}}{=} \{f \in \mathcal{F} \mid f(0) = 0\}$$

Trois opérateurs sur les fonctions sont utilisés, la convolution (*), la déconvolution (\otimes) et la clôture sous-additive (\cdot^*).

$$(f * g)(t) \stackrel{\text{def}}{=} \inf_{0 \leq s \leq t} f(t-s) + g(s) \quad (f \otimes g)(t) \stackrel{\text{def}}{=} \sup_{s \geq 0} f(t+s) - g(s)$$

$$f^* \stackrel{\text{def}}{=} \delta_0 \wedge f \wedge (f * f) \wedge (f * f * f) \wedge \dots$$

2.2 Bases de calcul réseau

En calcul réseau, on modélise un *flux* par sa courbe de trafic cumulé : $A \in \mathcal{F}_0$, où $A(t)$ représente le nombre de bits émis par le flux sur l'intervalle $[0, t)$. Par convention, $A(0) = 0$.

Mais A représente le flux "réel", ce qui va transiter sur le réseau, et qui est inconnu au moment de l'analyse. En pratique, on va travailler avec une abstraction du flux, un contrat de trafic, appelé « courbe d'arrivée ». On dit que A admet une courbe α pour courbe de service (noté $A \prec \alpha$) ssi

$$\forall t, s \in \mathbb{R}_{\geq 0} : A(t+s) - A(t) \leq \alpha(s) \iff A \leq A * \alpha \quad (1)$$

On voit qu'il s'agit d'une sur-approximation : on cherche une enveloppe pour un phénomène réel A , et plusieurs pourront convenir, capturant plus ou moins précisément le réel.

Un *serveur* S va être une application de \mathcal{F} dans \mathcal{F} , qui associe à un flux d'entrée un flux de sortie. De même que pour les flux, on ne manipule pas directement les serveurs, mais un contrat de service. On dit qu'un serveur S offre un service de courbe β ssi

$$\forall A, A' : A \xrightarrow{S} A' \implies A' \geq A * \beta \quad (2)$$

On peut borner le délai subi par un flux dans un serveur par la déviation horizontale $h(\alpha, \beta)$ entre la courbe d'arrivée α et la courbe de service β , comme illustré sur la figure 1. On y retrouve quelques intuitions du monde réel si la courbe d'arrivée croît plus vite que la courbe de service, cela signifie que le système est surchargé. Il est important d'arriver à modéliser les phénomènes au plus juste, c'est à dire, pour les courbes d'arrivée, d'avoir la courbe la plus petite possible.

Une propriété particulièrement intéressante pour cet article est le résultat suivant : soit A un flux, alors $A \otimes A$ est la meilleure courbe d'arrivée possible pour le flux A^1 . Par monotonie de la déconvolution, si on a deux bornes A^M et A^m telles que $A^m \leq A \leq A^M$, alors, $A^M \otimes A^m \geq A \otimes A$ est une courbe d'arrivée de A .

$$A^m \leq A \leq A^M \implies A \prec A^M \otimes A^m \quad (3)$$

Le principe de notre démarche est d'arriver, par des méthodes d'ordonnement, à construire un encadrement (A^m, A^M) afin de calculer une courbe d'arrivée.

¹ Formellement, si α est une courbe d'arrivée pour A , alors $\alpha \geq A \otimes A$.

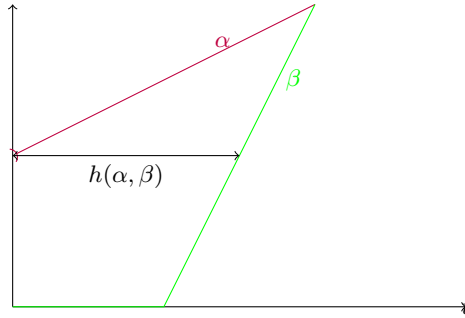


Fig. 1. Exemple de courbe d'arrivée, de service et déviation horizontale

2.3 Gains attendus pour le calcul réseau

Prenons ici un exemple simplifié pour illustrer le gain attendu et le principe de la méthode. Considérons un système périodique de période T . Dans chaque période, b bits sont émis. Si l'on ne connaît rien sur les moments d'émission de ces messages, on doit considérer les pires cas, dont celui où toutes les données sont émises en début de période, ce qui conduit à la courbe d'arrivée A de la figure 2. De plus, certains auteurs préfèrent éviter les courbes en escalier, et manipulent en fait la sur-approximation linéaire γ . Si on sait par contre que les données sont émises de façon régulière, un message de $\frac{b}{5}$ bit étant émis tous les $\frac{T}{5}$ unités de temps par exemple, on obtient la courbe en escalier B , beaucoup plus lisse. Supposons que le service soit celui de la figure β , il apparaît clairement que la borne de délai (la déviation horizontale entre courbe d'arrivée et de service) calculée pour le flux B est trois fois plus petite que celle du flux A .

2.4 Des courbes de trafic aux courbes d'arrivée

Nous ne présenterons pas dans cet article, par manque de place, le détail technique du passage des courbes de trafic (A^m, A^M) calculées au paragraphe 3 aux courbes de service. Deux points sont à traiter. Premièrement, il s'agit de calculer une déconvolution entre courbes en escalier. Nous pourrions reprendre les algorithmes généraux de [1], mais dans ce cas particulier, une résolution directe est aussi simple et plus rapide à implanter. Le second point consiste à étudier le passage d'une étude d'ordonnancement sur un horizon fini (une hyper-période) à une courbe d'arrivée à horizon infini.

3 Méthode d'ordonnancement pour le calcul de courbe d'arrivée

Comme présenté dans l'équation (3), le but des méthodes à base d'ordonnancement est de calculer un encadrement (A^m, A^M) du flux de données produits par un ensemble d'applications.

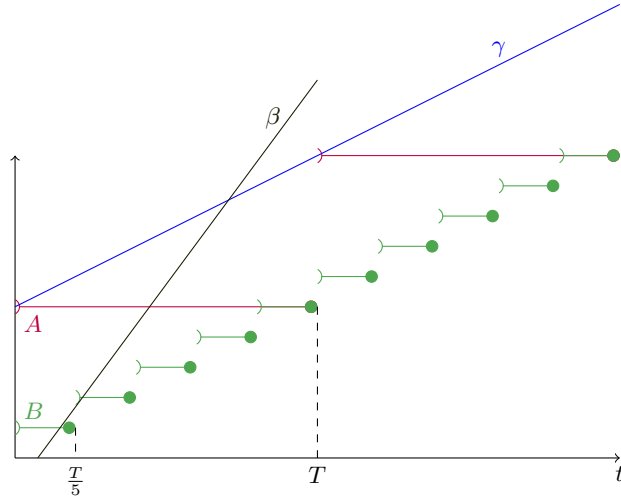


Fig. 2. Trafics plus ou moins lissés

Le modèle de tâches que nous allons considérer est le suivant : des tâches périodiques, avec un ordonnancement à priorités fixes. Chaque tâche τ émet un unique message de taille variable en fin d'exécution². Comme on va devoir étudier les émissions de message au plus tôt et au plus tard, on a besoin d'avoir le meilleur et le pire temps d'exécution.

Si on sait calculer ainsi les dates de fin d'exécution au plus tôt et au plus tard, on peut encadrer le débit réel : en effet, la courbe de trafic cumulé la plus grande A^M correspond au cas où les tâches finissent le plus tôt et émettent les plus grandes trames ; inversement, la courbe de trafic cumulé la plus petite correspond aux exécutions au plus tard et aux données les plus petites.

Ainsi, chaque tâche τ_i sera caractérisée par sa date de réveil R_i , sa priorité P_i (en supposant une unique tâche par priorité), son meilleur (resp. pire) temps d'exécution C_i^m (resp. C_i^M), sa période T_i et son échéance D_i . Le message émis est compris entre une taille minimale S_i^m et une maximale S_i^M .

Nous notons respectivement Θ_i^m la date d'émission au plus tôt, et Θ_i^M celle au plus tard. L'ensemble des tâches de plus haute priorité que τ_i est noté $hp(\tau_i)$:

$$hp(\tau_i) = \{\tau_j \mid P_j < P_i\}$$

Pour simplifier l'étude, nous supposons aussi que :

Hyp. 1 les tâches sont indépendantes (par de relation de précédence, pas de ressource partagée) ;

² C'est une hypothèse réaliste dans le monde temps réel embarqué, où une tâche périodique commence généralement par lire des données d'état, puis calcule une information, et émet le résultat de ses calculs en fin d'exécution. Si une tâche émet plusieurs message en fin d'exécution, cela revient du point de vue du débit à un seul message dont la taille est la somme des tailles individuelles.

Hyp. 2 des date de réveil nulles³ ($R_i = 0$);

Hyp. 3 des échéances plus petites que les périodes ($D_i \leq T_i$).

Avec les hypothèses 2 et 3, nous pouvons borner l'étude sur l'hyper-période $T = ppcm(T_i)$ [11]. Et l'hypothèse 2 implique que le pire (resp. meilleur) temps d'exécution est égal au pire (resp. meilleur) temps de réponse Θ_i^m (resp. Θ_i^M).

3.1 Évaluation du bag

La méthode classique pour calculer une courbe d'arrivée est de diviser la taille maximale des messages par la durée minimale entre deux émissions (bag). Le délai maximal (resp. minimal) entre deux émissions du même message correspond à l'intervalle de temps entre la fin au plus tôt (resp tard) d'une instance et sa terminaison suivante au plus tard (resp. tôt). La figure 3 illustre ces deux situations.

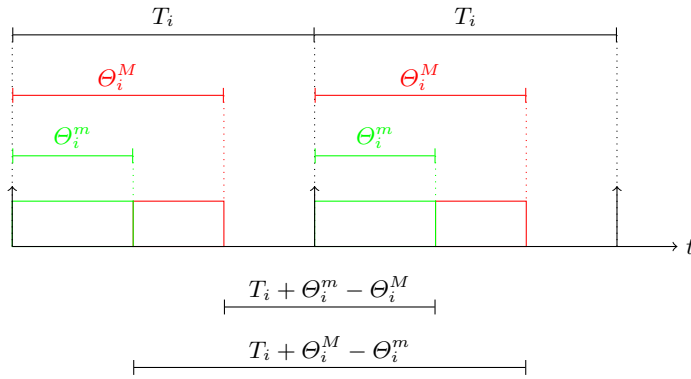


Fig. 3. Dédution du bag à partir des informations issues de l'étude d'ordonnancement

Nous pouvons en déduire un valeur optimale (i.e. maximale) pour bag_i :

$$bag_i = T_i + \Theta_i^m - \Theta_i^M \quad (4)$$

3.2 Approximation des traffics cumulés réels utilisant l'ordonnancement des tâches

Le pire temps de réponse [7,9,12] de la tâche τ_i dépend de son pire temps d'exécution C_i^M et des interactions des tâches plus prioritaires ($hp(\tau_i)$). L'ex-

³ Des travaux non encore publiés montrent que les dates non nulles répartissent la charge dans l'hyper-période, rendent le système encore plus lisse, et améliorent le gain de la méthode.

pression mathématique du pire temps de réponse est la suivante :

$$\Theta_i^M = C_i^M + \sum_{j \in hp(\tau_i)} \left\lceil \frac{\Theta_i^M}{T_j} \right\rceil \cdot C_j^M \quad (5)$$

Le meilleur temps de réponse est la notion duale au pire temps de réponse. Il est calculé en considérant que la tâche τ_i a le temps d'exécution le plus court et qu'elle ne subit aucune interaction de la part des instances plus prioritaires.

$$\Theta_i^m = C_i^m \quad (6)$$

À l'aide du calcul du pire et meilleur temps de réponse d'une tâche nous pouvons déterminer une approximation du trafic cumulé minimal et maximal (A^m et A^M). La courbe A^m (resp. A^M) est définie par une fonction en escalier correspondant à l'émission du message de taille minimale S_i^m (resp. maximale S_i^M) au plus tard Θ_i^M (resp. tôt Θ_i^m); ce motif se répétant à chaque période de la tâche. La figure 4 illustre la construction des fonctions A^m et A^M .

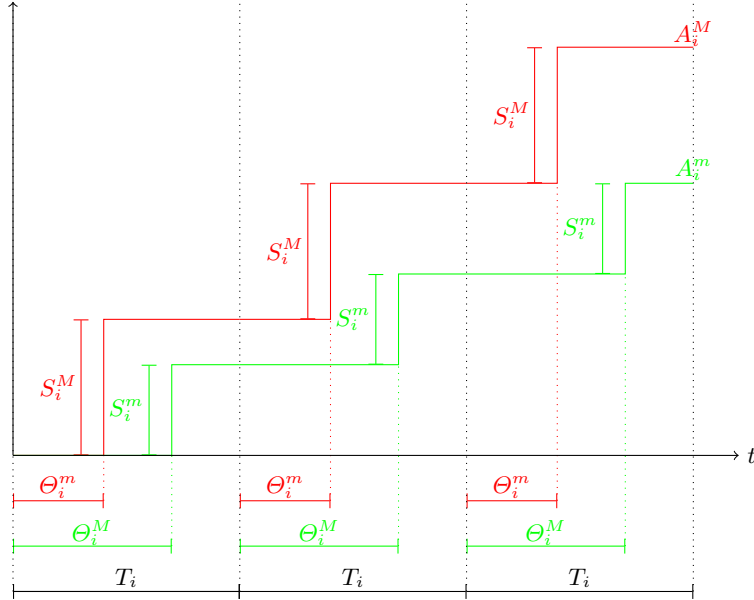


Fig. 4. Représentation graphique des fonctions A^m et A^M .

L'expression mathématique des fonctions A_i^m et A_i^M pour une tâche τ_i est donnée par les deux équations suivantes :

$$A_i^m(t) = \left\lfloor \frac{t}{T_i} \right\rfloor \cdot S_i^m + 1_{t - \lfloor \frac{t}{T_i} \rfloor \cdot T_i > \theta_i^m} \cdot S_i^m \quad (7)$$

$$A_i^M(t) = \left\lfloor \frac{t}{T_i} \right\rfloor \cdot S_i^M + 1_{t - \lfloor \frac{t}{T_i} \rfloor \cdot T_i > \theta_i^M} \cdot S_i^M \quad (8)$$

avec 1_x la fonction de test qui vaut 1 si son argument est vrai et 0 sinon.

L'approximation des trafics cumulés minimaux et maximaux est défini de la manière suivante :

$$A^m(t) = \sum_{\tau_i} A_i^m(t) \quad (9)$$

$$A^M(t) = \sum_{\tau_i} A_i^M(t) \quad (10)$$

3.3 Meilleurs et pires temps de réponse des instances

L'expérimentation de la méthode « par tâche » présentée au paragraphe précédent sur de petits exemples nous apparue un peu décevante⁴. Après réflexion, il apparut qu'en ne considérant que les tâches on génère un trafic qui fait comme si toutes les instances d'une même tâche pouvaient émettre en même temps dans une hyper-période. Nous avons donc développé une méthode qui s'intéressent aux instances de tâches ($\tau_{i,j}$ est la $j^{\text{ième}}$ instance de la tâche τ_i).

Notations Introduisons quelques notations nécessaires utiles à la compréhension de cette partie.

$[a, b]$ désigne une partie fermée de \mathbb{R} , \cup l'union, \setminus la différence, $\sup(\mathcal{X})$ la limite supérieure de \mathcal{X} et $\bar{\mathcal{X}}$ la fermeture de \mathcal{X} . Nous définissons aussi trois opérateurs :

1. l'opérateur d'élargissement d'une partie fermée de \mathbb{R} : \triangleright défini par

$$\mathcal{X} \triangleright \delta \equiv \mathcal{X} \cup [\sup(\mathcal{X}), \sup(\mathcal{X}) + \delta] \quad (11)$$

2. la longueur d'un sous-ensemble de \mathbb{R} définie comme l'intégrale de la fonction de test :

$$l(\mathcal{X}) \equiv \int_{\mathcal{X}} 1_{x \in \mathcal{X}} \quad (12)$$

Exemples : $l([2, 3]) = 1$, $l([2, 3] \cup [10, 12]) = l([2, 3]) + l([10, 12]) = 3$

3. la différence non vide de deux parties fermées de \mathbb{R}

$$\mathcal{X} \ominus \mathcal{Y} \equiv \overline{(\mathcal{X} \setminus \mathcal{Y})} \cup [\sup(\mathcal{X}), \sup(\mathcal{X})] \quad (13)$$

⁴ On trouvera au paragraphe 4.1 un petit cas d'étude illustratif.

La période d'activité [10] d'une instance $\tau_{i,j}$ est la durée entre sa date de réveil $R_{i,j}$ et la date de la fin de son exécution. Selon les notations précédentes la pire (resp. la meilleure) période d'activité de l'instance $\tau_{i,j}$ notée $\mathcal{I}_{i,j}^M$ (resp. $\mathcal{I}_{i,j}^m$) est définie par

$$\mathcal{H}_{i,j}^M = \bigcup_{\tau_{k,l} \in hp(\tau_{i,j})} \mathcal{I}_{k,l}^M \quad (14)$$

Périodes d'activité des instances Dans ce paragraphe, nous introduisons une nouvelle méthode pour calculer les pires et les meilleures dates de fin d'exécution d'instance. Cette technique consiste à calculer le pire (resp. le meilleur) période d'activité de chaque instance, puis à déterminer son pire (resp. le meilleur) temps d'exécution $\Theta_{i,j}^M = \sup(\mathcal{I}_{i,j}^M)$ (resp. $\Theta_{i,j}^m = \sup(\mathcal{I}_{i,j}^m)$).

L'algorithme suivant calcule la pire période d'activité de l'instance $\tau_{i,j}$:

Étape 1 Calculer la première approximation de période d'activité en prenant en compte uniquement l'exécution de l'instance, sans aucune interaction avec d'autres.

$$\mathcal{I}_{i,j}^{M,0} = [R_{i,j}, R_{i,j} + C_{i,j}^M] \quad (15)$$

Étape 2 Calculer l'approximation $\mathcal{I}_{i,j}^{M,n+1}$ en utilisant $\mathcal{I}_{i,j}^{M,n}$. Le principe consiste à appliquer l'opérateur d'élargissement (\triangleright) afin d'ajouter du temps de calcul utile pour que l'instance puisse finir son exécution sans interaction ($C_{i,j}^M - l(\mathcal{I}_{i,j}^{M,n})$). Finalement les interactions des instances plus prioritaires sont ajoutées à l'aide de l'opérateur de différence non-vide.

$$\mathcal{I}_{i,j}^{M,n+1} = \left(\mathcal{I}_{i,j}^{M,n} \triangleright \left(C_{i,j}^M - l(\mathcal{I}_{i,j}^{M,n}) \right) \right) \ominus \mathcal{H}_{i,j} \quad (16)$$

Étape 3 Déterminer si l'approximation de la période d'activité est le résultat.

- Si la date de fin d'exécution de l'instance dépasse son échéance, alors le système est non-ordonnançable.

$$\sup(\mathcal{I}_{i,j}^{M,n+1}) > D_{i,j} \quad (17)$$

- L'approximation correspond à la période d'activité effective si et seulement si :

$$\mathcal{I}_{i,j}^{M,n+1} = \mathcal{I}_{i,j}^{M,n} \quad (18)$$

- Sinon retourner à l'étape 2.

La figure 5 illustre le calcul de la période d'activité de l'instance $\tau_{3,1}$. La première trace d'exécution représente la première approximation de la période d'activité (i.e. l'exécution de l'instance sans interaction). La seconde montre la période d'activité des instances plus prioritaires $\mathcal{H}_{i,j}^M$. La trace suivante, met en évidence les effets des instances de plus fortes priorités. La dernière trace d'exécution montre l'utilisation de l'opérateur d'élargissement.

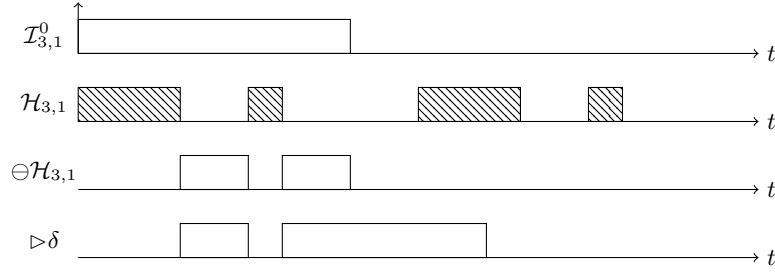


Fig. 5. Calcul de la période d'activité

3.4 Approximation des traffics cumulés réels utilisant l'ordonnancement des instances

A l'aide des pires périodes d'activité des instances nous en déduisons la meilleure approximation du trafic cumulé produit par l'instance $\tau_{i,j}$:

$$A_{i,j}^m(t) = 1_{t > \theta_{i,j}^M} \cdot S_i^m \quad (19)$$

L'approximation du trafic cumulé de la tâche τ_i correspond à la somme des flux produits par ses instances.

$$A_i^m(t) = \sum_{j=1}^{\infty} A_{i,j}^m(t) \quad (20)$$

Par conséquent les approximations des traffic cumulés minimal et maximal de la tâche τ_i sont définis de la manière suivante :

$$A_i^m(t) = \left\lfloor \frac{t}{T} \right\rfloor \cdot \frac{T}{T_i} \cdot S_i^m + \sum_{j=1}^{\frac{T}{T_i}} 1_{t > \theta_{i,j}^M} \cdot S_i^m \quad (21)$$

$$A_i^M(t) = \left\lfloor \frac{t}{T} \right\rfloor \cdot \frac{T}{T_i} \cdot S_i^M + \sum_{j=1}^{\frac{T}{T_i}} 1_{t > \theta_{i,j}^m} \cdot S_i^M \quad (22)$$

4 Exemples

Pour illustrer cette méthode, trois exemples seront présentés : un premier illustre la méthode sur un petit exemple (section 4.1) ; un second illustre le gain sur la taille de rafale, ainsi que l'effet de la variation du temps d'exécution et du nombre de tâches ; un troisième présente des résultats à partir de configuration réalistes générées aléatoirement.

4.1 Études de cas illustrative

Dans ce paragraphe, nous présentons une étude de cas simples à des fins d'illustration. Prenons un système temps réel composé de quatre tâches.

Afin de faciliter l'étude et l'analyse des résultats, certaines simplifications ont été faites : le meilleur et le pire temps d'exécution sont les mêmes pour de petites tâches ($C_i^m = C_i^M$), la taille des messages est constante ($S_i^m = S_i^M$). Le tableau suivant montre les paramètres des tâches et du flux :

Task	P_i	C_i^m	C_i^M	T_i	D_i	S_i^m	S_i^M
τ_1	1	1	1	10	10	10	10
τ_2	2	1	1	10	10	10	10
τ_3	3	2	3	20	20	60	60
τ_4	4	2	5	20	20	70	70

La figure 6 représente les courbes d'arrivée. La courbe rouge est obtenue avec la technique "classique" du calcul réseau : à partir de tailles de trame et du bag, on déduit la pire rafale et le débit long terme (cf paragraphe 2.3). La courbe bleue utilise une technique (non présentée ici) qui ne prend pas en compte les instances de tâche mais seulement les tâches. La courbe verte est calculée en prenant en compte le comportement des instances de tâches. Les trois courbes noires correspondent à trois charges réseaux différentes envisagées dans cet exemple.

On voit clairement le gain de la nouvelle méthode : le trafic généré est beaucoup plus lisse, et la pire rafale est bien plus petite. En ce qui concerne les délais (déviations horizontales entre la courbe d'arrivée et de service), le gain dépend bien sûr non pas seulement de la courbe d'arrivée mais aussi de celle du service. Mais là aussi, les gains sont conséquents, même s'ils diminuent avec la charge (en effet, sur un système peu chargé, c'est surtout les rafales qui génèrent le délai).

4.2 Efficacité

L'objectif de ce paragraphe est d'évaluer l'avantage des techniques proposées en fonction de plusieurs paramètres : le nombre de tâches et la variation du temps de calcul. Dans cette étude, la technique basée sur les comportements des instances est la seule considérée.

La notion d'*efficacité* introduite ici pour caractériser le gain de notre technique est calculée comme suit :

$$efficiency = \frac{\text{rafale classique} - \text{rafale par instance}}{\text{rafale classique}} \quad (23)$$

Pour chaque couple $(n, \Delta) \in [1, 100] \times [0, 1]$, nous générons cent configurations. Dans chaque configuration, les tâches $(\tau_i)_{i \in [1, n]}$ sont générées, avec la même période T , un délai d'exécution maximal C_i^M choisi au hasard entre 1 et $\frac{T}{n}$ et une durée minimale $C_i^m = \Delta C_i^M$.

L'expérimentation met en évidence deux points :

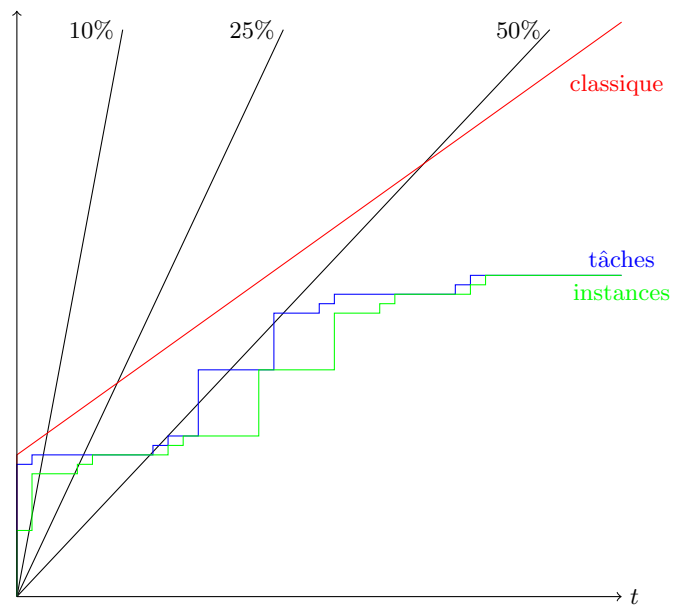


Fig. 6. Exemple

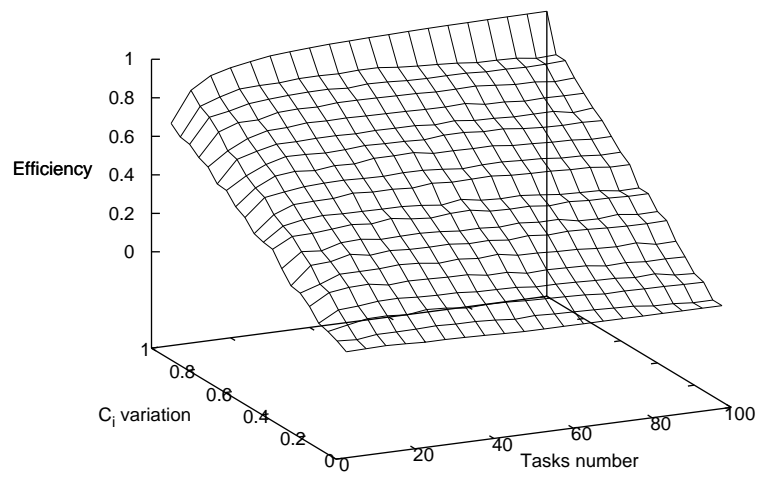


Fig. 7. Efficacité de la méthode

1. L'efficacité augmente rapidement avec le nombre de tâches (presque égale 1 avec 20 tâches). En effet, la technique classique estime que toutes les tâches peuvent produire leurs messages en même temps. Cette approximation est très pessimiste. Avec notre méthode, plus il y a de tâches, moins elles peuvent émettre simultanément.
2. L'efficacité décroît rapidement avec la variation du temps d'exécution des tâches. Ce résultat est dû au fait que, lorsque C^m/C^M tend vers 0, le temps d'exécution C^m tend aussi vers 0, et toutes les instances peuvent effectivement se terminer au même instant 0.

4.3 Exemple réaliste

Nous allons faire le même genre d'analyse que dans les sections 4.1 et 4.2, c'est à dire évaluer le gain de la méthode pour trois charges réseau (10 %, 25 %, 50 %), mais avec des configurations réalistes, en augmentant progressivement la taille de la configuration étudiée.

Dans chaque configuration, n tâches sont générées (avec n un multiple de trois), également réparties en trois classes (chacune de taille $n/3$). Dans chaque classe, le pire temps d'exécution de chaque tâche τ_i est $C_i^M = \max(\lfloor \frac{T_i}{n} \rfloor, 1)$ et la taille du message est constante, égale à la période.

La première classe, "petite et forte priorité", se compose de tâches τ_i avec une période T_i choisie au hasard dans $\{10, 20, 30, 40, 50\}$, et un ratio $\frac{C^m}{C^M} = 0,9$.

Dans la deuxième classe "moyenne", les périodes sont choisies au hasard dans $\{50, 100, 150, 200, 250\}$ et le ratio $\frac{C^m}{C^M} = 0,75$. La troisième classe de priorité "grande et faible priorité" a des périodes dans $\{300, 400, 500\}$ et un ratio $\frac{C^m}{C^M} = 0,5$.

Ces classes font des hypothèse plutôt réalistes pour les systèmes temps-réel, où les tâches à haute fréquence font quelques petits calculs, générant de petits messages urgents avec une priorité élevée, souvent pour les fonctions de contrôle/commande, alors que de tâches de priorité plus faibles effectuent des tâches de fond, comme la supervision, l'enregistrement de l'activité, et génèrent de plus gros messages.

Pour chaque valeur de n dans $[3, 30]$, 100 configurations ont été générées, et les gains moyens sont tracés dans la figure 8. Sont tracés le gain pour la rafale (*burst*) et le gain en délai pour différentes charges. Comme prévu, le gain diminue avec la charge du réseau. Mais les résultats restent proches. Mais le principal résultat est que, avec des configurations réalistes, le gain augmente avec la taille des systèmes, croît très rapidement au dessus de 50 %, et atteint près de 80 % avec 30 tâches.

5 Conclusion

Dans cet article, nous avons fait collaborer deux méthodes du monde temps réel, l'ordonnancement et le calcul réseau, en demandant à l'ordonnancement de prendre en compte des mécanismes que le calcul réseau ne peut pas traiter.

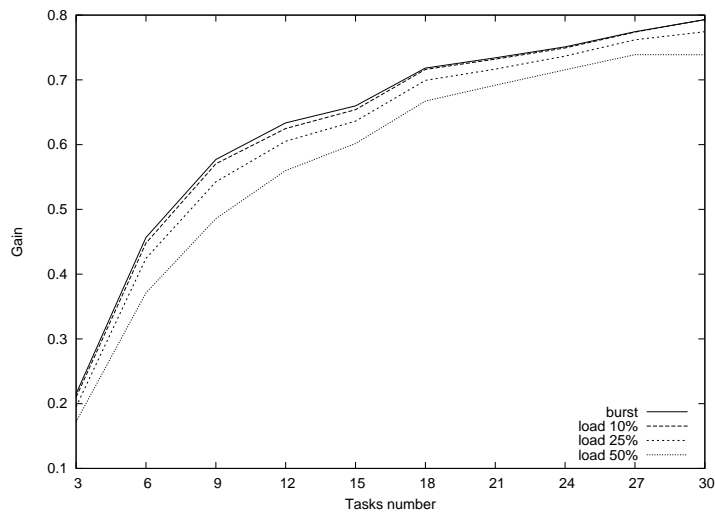


Fig. 8. Gain avec des configurations réalistes

Les premiers résultats sont extrêmement prometteurs, d'autant qu'ils grandissent avec le nombre de tâches du système.

Références

1. Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 17(4), october 2007. <http://www.springerlink.com/content/876x51r6647r8g68/>.
2. Marc Boyer, Laurent Jouhet, and Anne Bouillard. Notations pour le calcul réseau. *Journal Européen des Systèmes Automatisés*, 43(7-9) :921-935, 2009. Actes du 7ème colloque francophone sur la Modelisation des Systemes Reactifs (MSR'09).
3. Cheng-Shang Chang. *Performance Guarantees in communication networks*. Telecommunication Networks and Computer Systems. Springer, 2000.
4. Rene L. Cruz. A calculus for network delay, part I : Network elements in isolation. *IEEE Transactions on information theory*, 37(1) :114-131, January 1991.
5. Rene L. Cruz. A calculus for network delay, part II : Network analysis. *IEEE Transactions on information theory*, 37(1) :132-141, January 1991.
6. Jérôme Grieu. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques*. PhD thesis, Institut National Polytechnique de Toulouse (INPT), Toulouse, Juin 2004.
7. M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5) :390-395, 1986.
8. Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus*, volume 2050 of LNCS. Springer Verlag, 2001. http://lrcwww.epfl.ch/PS_files/NetCal.htm.

9. J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm : exact characterization and average case behavior. *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171, 1989.
10. JP Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209, 1990.
11. Joseph Y. T. Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4), 1982.
12. K. Tindell. An extendible approach for analyzing fixed priority hard real-time tasks. Technical Report YCS189, 1992.