Robustesse des applications temps-réels multicoeurs

Yves Mouafo Tchinda, Annie Choquet-Geniet, Gaëlle Largeteau

Robustesse des applications temps-réels multicoeurs

Yves MOUAFO
LIAS-ENSMA
86961 Futuroscope Chasseneuil
yves.mouafo@ensma.fr

Annie CHOQUET-GENIET
LIAS-ENSMA
86961 Futuroscope Chasseneuil
annie.geniet@ensma.fr

Gaëlle LARGETEAU-SKAPIN
SXlim-SIC Universié de Poitiers,
86962 Futuroscope Chasseneuil
gaelle.largeteau.skapin@univ-poitiers.fr

***Abstract*—Ce travail traite de la tolérance aux pannes des applications temps réel s'exécutant dans un environnement multicoeurs où une panne peut survenir à tout instant sur l'un des coeurs du processeur. Il est alors crucial de réorganiser les tâches sur les coeurs restants afin d'assurer que l'ordonnancement reste valide malgré la panne. Plusieurs idées sont à envisager à cet effet, le présent article en expose une. Il s'intéresse en particulier aux systèmes de tâches périodiques, indépendantes, à départs simultanés et à échéances sur requête, ordonnancés par un algorithme équitable. Notre approche est basée sur une redondance matérielle limitée (un coeur en plus) et une réconfiguration après la panne des tâches dont les échéances ont été initialement contraintes. Ces échéances sont calculées en répartissant de façon équitable les temps creux de l'ordonnancement sur l'hyperpériode.**

I. INTRODUCTION

De nombreux équipements embarquent des applications temps-réel, offrant aux utilisateurs divers services dans des domaines variés, à l'instar de l'aéronautique, l'automobile, les télécommunications, l'électroménager. On qualifie de temps-réel, toute application informatique dont l'exactitude dépend aussi bien de la validité des résultats que du temps mis pour les obtenir [1]. De telles applications sont constituées de tâches, que nous supposons périodiques et indépendantes. Une tâche se caractérise par sa date de réveil r_i, sa pire durée d'exécution C_i, sa période T_i et son délai critique D_i. Nous supposons que les tâches sont à départs simultanés (les r_i sont égaux) et à échéances sur requêtes ($D_i = T_i$).

De nos jours, l'évolution des besoins , l'intégration croissante des fonctionnalités critiques et l'évolution des composants mettent en évidence l'étroitesse des ressources (mémoire et CPU) nécessaires à l'exécution de systèmes [2]. C'est pourquoi les équipements sont souvent dotés de plusieurs unités de traitement (plusieurs processeurs ou un processeur à plusieurs coeurs), posant ainsi le problème de l'ordonnancement multiprocesseur. On distingue deux approches d'ordonnancement multiprocesseur [3] : l'approche globale où il y a une seule file d'attente commune à tous les coeurs et l'approche partitionne où chaque coeur dispose de sa propre file d'attente. Sous nos hypothèses (périodicité et indépendance), une condition nécessaire et suffisante d'ordonnançabilité d'un système de *n* tâches sur *m* unités de traitement est:

$$U = \sum_{i=1}^{n}(C_i/T_i) \leq m).$$

Nous considérons un ordonnancement suivant une approche globale par un algorithme équitable. Un algorithme est dit équitable si, à tout instant, l'écart absolu entre l'ordonnancement idéal (la charge à exécuter est proportionnelle à $\frac{C_i}{T_i}$) et l'ordonnancement réel pour chaque tâche est inférieur au quantum de temps [4]. Chaque tâche τ_i de charge $U_i = C_i/T_i$ est subdivisée en C_i sous-tâches de durées d'exécution égale au quantum. Les formules suivantes [5] permettent de calculer respectivement la pseudo-date de réveil r_i^j et la pseudo-échéance d_i^j de la j^e sous-tâche d'une tâche τ_i :

$$r_i^j = \lfloor \ \frac{j}{U_i} \ \rfloor; \ d_i^j = \lceil \ \frac{j+1}{U_i} \ \rceil; (j \geq 0) \ ^1.$$

Dans ce travail, nous avons choisi PD2 comme algorithme d'ordonnancement.

Au cours de l'ordonnancement, à n'importe quel instant, une unité de traitement peut subir une panne, affectant la sous-tâche qui s'y exécutait. Nous supposons l'existence d'un mécanisme de détection instantanée de cette panne. Se pose alors la question de la réorganisation du système pour que les tâches poursuivent leur exécution sur les coeurs restants tout en respectant leurs échéances et les contraintes d'équité.

L'approche que présente cet article suggère une action aux niveaux matériel et systémique. Au niveau matériel, elle prévoit l'ajout d'un $(m+1)^e$ coeur. Au niveau systémique, on prévoit de contraindre les échéances des tâches au début de l'ordonnancement et de les relâcher après la panne. Une telle action soulève une préoccupation: comment calculer les échéances des tâches? L'idée consiste à répartir équitablement entre les tâches la durée totale des temps creux de l'ordonnancement sur l'hyperpériode.

La suite de ce document est organisée en quatre sections. La prochaine section (*section 2*) présente l'approche dans son ensemble. Elle est suivie d'une illustration sur un exemple (*section 3*), puis des résultats expérimentaux (*section 4*). Enfin, des perspectives sont dégagées (*section 5*) et nous terminons par une conclusion (*section 6*).

II. L'APPROCHE

Soit un système $S = (\tau_1(C_1, T_1), ..., \tau_n(C_n, T_n))$ de *n* tâches périodiques, indépendantes, à départs simultanés et à échéances sur requêtes, ordonnançable sur un processeur à *m* coeurs par l'algorithme PD2. Nous avons donc U=$\leq$ m.

Notre approche préconise une action aux niveaux matériel et système. Au niveau matériel, il s'agit d'introduire une

[1] $\lfloor$ x $\rfloor$ et $\lceil$ x$\rceil$ désignent respectivement, la partie entière inférieure et la partie entière supérieure du nombre x

redondance en exécutant le système S sur un processeur à *(m+1)* coeurs. Ainsi, l'ordonnançabilité du système reste garantie en cas de défaillance d'un des coeurs. Nous avons démontré (publication à venir) que dans une telle configuration, si un coeur tombe en panne et que la sous-tâche de l'instance de la tâche qui s'y exécutait est négligée, le système reste valide et équitable. Dans le cas contraire (reprise obligatoire de la sous-tâche), comment garantir la validité et l'équité?

Au niveau du système, dans un premier temps, un système de tâches à échéances contraintes

$$S' = (\tau'_1(C_1, D'_1, T_1), ..., \tau'_n(C_n, D'_n, T_n))$$

est construit à partir de S. Comme l'illustre la figure 1 ci-dessous, cela permet de prévoir pour chaque tâche une *fenêtre de tolérance* d'au moins une unité de temps pour reprendre l'exécution de la sous-tâche affectée par la panne. Une conséquence immédiate est la réduction des fenêtres d'exécution des sous-tâches.

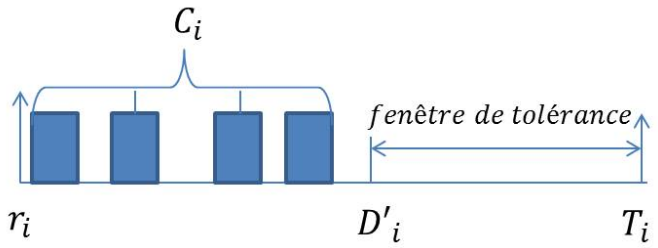

Fig. 1. fênêtre de tolérance d'une tâche

Au départ, *S'* s'exécute sur le processeur à *m+1* coeurs. Lorsque la panne survient, une reconfiguration est opérée sur les sous-tâches des instances non exécutées. On obtient au final un système *S"* dont les paramètres des sous-tâches sont ceux de *S'* avant l'instant de la panne et de *S* après. La figure 2 ci-dessous illustre la démarche.

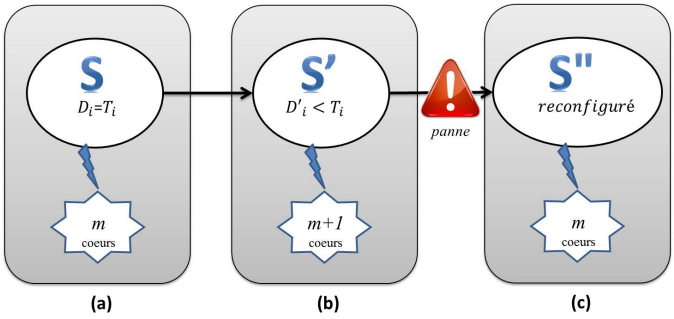

Fig. 2. démarche de tolérance aux pannes

Dans la suite de cette section, nous présentons les méthodes de calcul de S' et S".

A. *Calcul d'échéances de S'(fig 2.b)*

L'idée est de répartir la durée *X* des temps creux de l'ordonnancement sur l'hyperpériode *H* entre les tâches comme marge de tolérance.
Pour un processeur à (m+1) coeurs, nous avons

$$X=((m+1)-U)*H,$$

où *U* est la charge du système.

Du fait de la redondance matérielle, il y au moins *H* temps creux exploitables par hyperpériode. Comme le système comporte *n* tâches, il revient à chacune une proportion de *X/n* temps creux à répartir entre ses instances. Pour trouver l'équivalent pour une instance, il suffit de diviser cette proportion par le nombre d'instances H/T_i de la tâche. D'où, la marge de tolérance d'une tâche sur sa période est : $\frac{X*T_i}{n*H}$. La taille de la fenêtre de tolérance doit être au moins égale à 1; l'échéance peut donc être obtenue par la formule suivante:

$$D'_i = T_i - MAX(1, \lfloor \tfrac{X*T_i}{n*H} \rfloor)$$

.

Les pseudo-dates de réveil et pseudo-échéances des sous-tâches du système S' sont calculées par les formules suivantes:

$$r_i^{'j} = \lfloor \tfrac{j}{CH_i} \rfloor; \; d_i^{'j} = \lceil \tfrac{j+1}{CH_i} \rceil;$$

où CHi désigne le facteur de charge de la tâche τ_i.

La figure 3 ci-dessous résume les étapes de calcul des délais critiques du système S'.

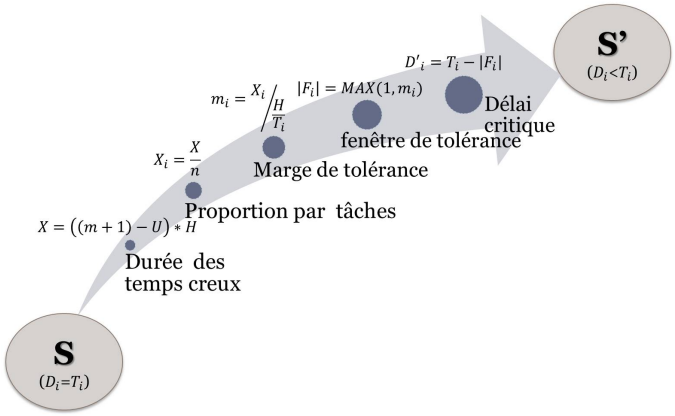

Fig. 3. Calcul du délai critique

B. *Réconfiguration: système S" (fig 2.c)*

Lorsqu'une panne survient à l'instant t_p, elle affecte une sous-tâche $\tau_{i_0}^{j_0}$ qui appartient à l'instance $k_0 = \lfloor \frac{j_0}{C_{i0}} \rfloor$ de la tâche τ_{i_0}. Les contraintes du système *S'* doivent être relâchées pour la suite de l'ordonnancement et l'exécution de cette sous-tâche doit être rattrapée.

La réconfiguration se passe de la manière suivante:

- tâches non affectées $\tau'_i(i \neq i_0)$:

retour aux paramètres du système S pour les instances non-encore exécutées

$$(r_i^{'j},d_i^{'j}) \longmapsto (r_i^j,d_i^j)\ ,\ i \neq i_0$$

- tâche affectée τ'_{i_0}**:**

$$j \geq k_0.C_{i0} \Longrightarrow (r_{i_0}^{'j},d_{i_0}^{'j}) \longmapsto (r_{i_0}^j,d_{i_0}^j);$$

$$j_0 < j < k_0 C_{i0} \Longrightarrow (r_{i_0}^{'j},d_{i_0}^{'j}) \longmapsto (r_{i_0}^{'j},d_{i_0}^{'j});$$

reprise de $\tau_{i_0}^{j_0}$:
$(r_{i_0}^{'j_0},d_{i_0}^{'j_0}) \longmapsto (D'_{i_0},T_{i_0})$.

III. Exemple illustratif

Soit à ordonnancer le système
$S = (\tau_1(2,3), \tau_2(2,6), \tau_3(6,8), \tau_4(3,8), \tau_5(5,12))$.

A. Calcul des délais critiques de S'(table1)

$U_s = 2,54 < 3; m = 3$ H=ppcm(3,6,8,12)=24
$X = ((m+1) - U) * H = 35$
Proportion par tâche: $X/n = 35/5 = 7$

TABLE I. Marge de tolérance par tâche

tâche	instance	marge	Délai critique
τ_1	8	$\lfloor 7/8 \rfloor = 0$	$3 - max(1,0) = 2$
τ_2	4	$\lfloor 7/4 \rfloor = 1$	$6 - max(1,1) = 5$
τ_3	3	$\lfloor 7/3 \rfloor = 2$	$8 - max(1,2) = 6$
τ_4	3	$\lfloor 7/3 \rfloor = 2$	$8 - max(1,2) = 6$
τ_5	2	$\lfloor 7/2 \rfloor = 3$	$12 - max(1,3) = 9$

On obtient le système contraint
$S' = (\tau'_1(2,2,3), \tau'_2(2,5,6), \tau'_3(6,6,8), \tau'_4(3,6,8),$
$\tau'_5(5,9,12))$ de facteur de charge
$CH_{S'} = \sum_{i=1}^{n}(C_i/D_i) = 3.45$.

Le tableau 2 ci-dessous é les paramètres des sous-tches des premières instances de chaque tche des systmes S et S'.

TABLE II. Sous-tâches de S et de S'

i	j	r_i^j	d_i^j	$r_i^{'j}$	$d_i^{'j}$
1	0	0	2	0	1
	1	1	3	1	2
2	0	0	3	0	3
	1	3	6	2	5
3	0	0	2	0	1
	1	1	3	1	2
	2	2	4	2	3
	3	4	6	3	4
	4	5	7	4	5
	5	6	8	5	6
4	0	0	3	0	2
	1	2	6	2	4
	2	5	8	4	6
5	0	0	3	0	2
	1	2	5	1	4
	2	4	8	3	6
	3	7	10	5	8
	4	9	12	7	9

B. Réconfiguration du système S' en S"

La figure 4 suivante présente les paramètres des sous-tâches des première et deuxième instances de τ_3 ainsi que la réconfiguration qui s'opère à la suite d'une panne survenue à l'instant 3 et qui affecte sa 3^e sous- tâche.

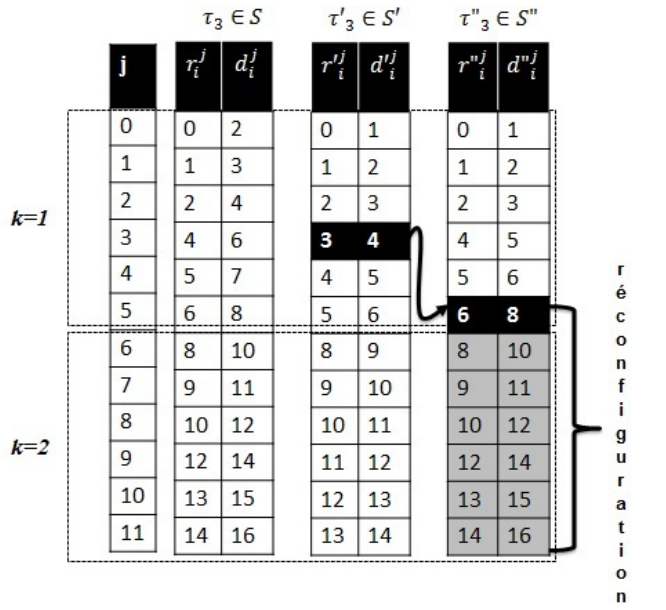

Fig. 4. Exemple de réconfiguration

On remarque que la sous-tâche affectée (j=3) de fenêtre [3,4] est rattrappée dans la fenêtre de tolérance [6,8]. Les sous-tâches des instances suivantes (k=2 et plus) ont retrouvés les fenêtres du système initial, tandis que les sous-tâches (j=4 et j=5) de l'instance en cours (k=1) gardent leurs fenêtres contraintes .

IV. Expérimentation

Afin de tester notre approche de tolérance, un prototype logiciel (FTA: Fault Tolerance Analyser) est en cours de conception. Des tests effectués sur le système S de l'exemple précédent montrent que, quelque soit l'instant auquel la panne survient et qu'importe le coeur affecté, le système reste valide et équitable. Peut-on alors généraliser l'approche? Pour se faire une idée, nous avons procédé à deux essais portant chacun sur 550 systèmes. Le premier essai porte sur 11 générations aléatoires de systèmes de tâches tels que $U \leq m$ (appelés *systèmes NOT FULL*) et le deuxième essai sur 11 générations aléatoires de systèmes $U \cong m$ (*systèmes FULL*). Chaque génération correspond à un taux de tâches lourdes ($U \geq \frac{1}{2}$) dans les systèmes et contient 50 systèmes. Ceci nous permet de voir si la charge du système et le nombre de tâches lourdes ont une influence sur le résultat.

Le premier essai enregistre une validité et équité de l'ordonnancement à 99.54 pour cent et le deuxième 96.61 pour cent. Les échantillons de systèmes non valides prélevés sont constitués essentiellement de systèmes FULL. Il en ressort alors que plus la charge du système se rapproche de *m*, moins l'ordonnancement a des chances d'être valide. Par ailleurs, il a été identifié des systèmes avec $D'_i < C_i$ ce qui revèle

une limite de la méthode et entraine la non validité de l'ordonnancement.

V. PERSPECTIVES

La tolérance aux pannes par l'exploitation des temps creux ci-dessus présentée soulève des interrogations qui orienteront nos futures investigations:
- Qu'est-ce qui explique la non-équité ou la non validité de l'ordonnancement de certains systèmes?
- Comment éviter d'avoir des systèmes avec $D'_i < C_i$?
- L'ordonnançabilité du système de tâches à échéances sur requête S sur *m* coeurs garantit-il celui du système contraint S' sur *m+1* coeurs? En d'autres termes,
$\sum_{i=1}^{n} \frac{C_i}{T_i} = m \Rightarrow \sum_{i=1}^{n} \frac{C_i}{D'_i} = m + 1$?
- Ne courrons-nous pas le risque de répartir plus de temps creux qu'il n'y en a dans l'hyperpériode?
$\sum_{i=1}^{n} MAX(1, \lfloor \frac{X * T_i}{n * H} \rfloor) \leq X$?
- Comment caractériser les tâches dont la marge de tolérance est nulle?
- Peut-on déduire une condition nécéssaire et suffisante d'ordonnançabilité?

Les réponses à ces questions serviront de base à l'établissement d'une preuve mathématique de notre approche.

VI. CONCLUSION

En définitive, ce travail soulève la problématique de l'ordonnanabilité d'un système temps réel soumis à une défaillance de l'un des coeurs du processeur sur lequel il s'exécute. Une approche de tolérance à une telle panne a été présenté. Elle consiste à exploiter les temps creux de l'ordonnancement pour déterminer une marge de tolérance qui servira de base à la construction d'une fenêtre de tolrance dans laquelle l'exécution de la sous-tâche impactée par la panne sera reprise. La taille de la fenêtre de tolérance est déduite de la période pour déterminer le délai critique d'une tâche.On passe ainsi d'un système de tâches à échéances sur requêtes à un système de tâches à échéances contraintes dont les fenêtres d'exécution des sous-tâches sont réduites. Ce système s'éxécute sur *m+1* coeurs; une fois la panne survenue, les contraintes sont levées et le système peut poursuivre son exécution sur les *m* coeurs restants avec des fenłtres d'exécution plus larges. Cette approche a été illustré sur un exemple et une expérimentation sur le prototype FTA a montré qu'elle assure la validité du système et l'équité dans la majorité des cas. Cependant des interrogations se posent en ce qui concerne l'ordonnançabilité du système contraint, la valeur du délai critique par rapport à la durée de la tâche et la durée totale de tolérance prévue qui peut excéder la durée des temps creux disponible. Dans la suite des travaux, des réponses à ces interrogations permettront d'affiner la méthode et de déterminer une condition nécessaire et suffisante d'ordonnançabilité. Si le délai critique calculé est inférieure à la pire durée d'exécution de la tâche, nous envisageons de considăréer cette durée comme délai critique. De plus nous étudierons si une répartition de la durée des temps creux proportionnellement à la charge de la tâche peut améliorer la validité et l'équité de l'ordonnancement.

REFERENCES

[1] J. Goosens, *Introduction à l'ordonnancement temps réel multiprocesseur*, Université Libre de Bruxelles

[2] T. Megel, *Placement, ordonnancement et mécanismes de migration de tâches temps-réel pour des architectures distribuées multicoeurs*, Thèse de doctorat de l'Université de Toulouse, avril 2012.

[3] M. Chéramy, A.M. Déplanche, P.E Hladik, *Ordonnancement temps réel: des politiques monoprocesseurs aux politiques muliprocesseurs*, Archives ouvertes, 2012.

[4] A.Geniet, G.Largeteau, A. Ouattara, *Mesures de l'équité d'une application temps-réel à l'aide de géométrie discrète*, LISI-ENSMA et SIC-Université de Poitiers, Journal Européen des Systèmes Automatisés, Hermès, 2009,43,pp.1065-1080

[5] S. MALO, *Contribution à l'analyse d'ordonnançabilité des applications temps-réel multiprocesseurs*,Thèse de doctorat ENSMA, décembre 2010.