Conception conjointe commande/ordonnancement et ordonnancement régulé

Daniel Simon

▶ **To cite this version:**

Daniel Simon. Conception conjointe commande/ordonnancement et ordonnancement régulé. École thématique. ÉCOLE D'ÉTÉ TEMPS RÉEL 2005 GdR Architecture, Réseaux et systèmes, Parallélisme (ARP) Thème Systèmes Temps-Réel et Qualité de Services (StrQdS) http://etr05.loria.fr/ Nancy, 13 - 16 Septembre 2005, 2005. inria-00000757

HAL Id: inria-00000757

https://cel.hal.science/inria-00000757

Submitted on 16 Nov 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Conception conjointe commande/ordonnancement et ordonnancement régulé

Daniel Simon
Inria Rhône-Alpes – Projet POP ART
655 avenue de l'Europe Montbonnot
38334 St Ismier Cedex
Daniel.Simon@inrialpes.fr

Abstract

Les performances de commande d'un système en boucle fermée dépendent aussi bien de l'algorithme utilisé que des paramètres d'exécution des programmes, telles que périodes, latences et gigue. Les tâches de commande sont soumises à des incertitudes temporelles dues à l'exécution des programmes sur la ressource de calcul. Conjointement à l'algorithme de commande les paramètres d'ordonnancement d'un programme de commande doivent être adaptés ou optimisés en vue d'améliorer les performances d'un contrôleur, en particulier sa robustesse temporelle. D'autre part les techniques de commande en boucle fermée peuvent être appliquées à l'ordonnanceur temps-réel lui même, le rendant ainsi adaptatif et robuste face aux incertitudes temporelles d'exécution. Cette démarche de conception conjointe et d'ordonnancement régulé est illustrée par quelques exemples.

1 Introduction : Commande et Rétroaction

Le but d'un système de commande est de contrôler un processus pour l'amener dans un état conforme aux désirs de l'utilisateur. Il est depuis longtemps apparu que la réalisation d'une commande en boucle fermée, où la valeur des sorties du processus rétroagit sur le signal de commande, permet d'améliorer les performances du système, exprimées par exemple en terme de précision de suivi de consignes ou d'insensibilité aux perturbations [15]. Le contrôleur réalisant la fonction de commande est le plus souvent réalisé sur un calculateur numérique suivant le schéma représenté par la figure 1. Les composants principaux du système sont :

- Le processus commandé : c'est le plus souvent un processus physique (machine électromécanique, processus chimique…) mais ce peut être aussi une entité informatique (serveur multimédia). Il est caractérisé par une dynamique d'entrée/sortie :

$$\begin{cases} \dot{X} = f(X,U,t) & \text{équation d'état} \\ Y = g(X,U,t) & \text{équation de mesure} \end{cases}$$

  où la sortie observée Y dépend de l'état interne X du système, de sa commande U et du temps t.
- Les capteurs permettent de mesurer les sorties (continues) du système. Elles sont échantillonnées (au moyen d'une horloge de période h)pour être transmises au calculateur numérique ;
- Les actionneurs reçoivent des commandes et agissent sur le processus. Les commandes calculées numériquement doivent être bloquées (le plus souvent en pratique par un bloqueur d'ordre zéro) pour pouvoir agir entre les instants d'échantillonnage ;
- Le régulateur K calcule les commandes en fonction de l'écart e entre la mesure Y et la consigne Y_d au moyen d'une fonction plus ou moins complexe et coûteuse. Il peut aussi inclure des fonctions de filtrage, de reconstruction d'état et de pré-compensation.

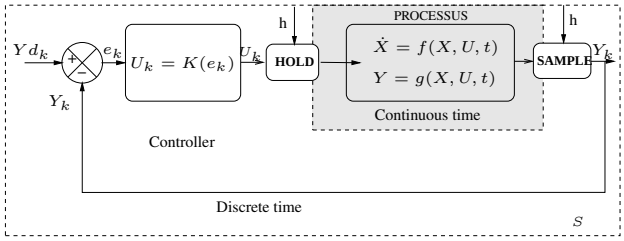

FIG. **1. Commande en boucle fermée**

Par rapport à une commande en boucle ouverte (sans rétroaction) les bénéfices attendus d'une commande en boucle fermée bien conçue sont multiples et permettent potentiellement d'améliorer les performances de commande sur plusieurs points :

- Stabilité (au sens entrée bornée/sortie bornée) : capacité à stabiliser un processus naturellement instable ;
- Augmentation de la précision en régulation et/ou en poursuite ;

– Accélération du temps de réponse sans sollicitations excessives des actionneurs ;
– Rejet de perturbations externes, mesurées ou non ;
– Robustesse aux incertitudes de modèle, garantissant un certain niveau de stabilité et de performances ;

Toutes ces performances ne peuvent être simultanément et arbitrairement améliorées, le concepteur du contrôleur doit gérer un compromis entre stabilité/précision/saturations/sensibilité/robustesse. En particulier une grande robustesse aux incertitudes de modèle du processus commandé se paye par un niveau de performance modéré en terme de temps de réponse et de bande passante (théorème du petit gain e.g.[43]).

Enfin, au delà des aspects purement algorithmiques, la réalisation d'une boucle de commande sur un calculateur numérique va perturber plus ou moins gravement ses performances par l'effet de l'échantillonnage et de retards induits.

Pour bénéficier des avantages de la commande fermée le contrôleur doit être correctement conçu, réalisé et paramétré. En revanche, une mauvaise conception peut entraîner une déstabilisation du système et un risque de divergence encore plus rapide qu'en boucle ouverte. Les divers aspects concernant l'algorithmique de commande et l'implémentation du contrôleur doivent être idéalement pris en compte simultanément pour tirer tous les bénéfices de l'approche boucle fermée.

2 Implémentation des systèmes de commande

Les systèmes de commande en boucle fermée sont pour la plupart périodiques, les entrées (lecture de capteurs), les calculs de commandes et les sorties (envoi de commandes aux actionneurs) sont effectués à une période unique et constante. Cependant, bien que la théorie classique de la commande des systèmes numérique repose sur l'utilisation d'un échantillonnage à période unique fixe, il a été mis en évidence, e.g. [3], que la réalisation d'un contrôleur sous forme multi-tâches et multi-cadences permet d'en améliorer les performances, même dans le cas de systèmes linéaires simples [4]. En effet, certaines parties d'un algorithme de commande (par exemple la mise à jour de paramètres) peuvent être moins urgentes à calculer que l'élaboration de la commande des modes rapides du système : leur calcul peut être sans dommage décalé dans le temps ou être exécuté à une cadence plus lente. En fait un système complexe est constitués de sous-ensembles aux dynamiques différentes devant être exécutées, puis coordonnées, en fonction de leurs caractéristiques temporelles respectives [44]. Le système de contrôle/commande doit donc gérer en parallèle (et en temps-réel) un certains nombre de contrôleurs élémentaires exécutés à leur cadence propre dans une hiérarchie de niveaux plus ou moins couplés.

Les applications de contrôle/commande, que l'on trouve par exemple en robotique, nécessitent d´exécuter des activités informatiques ayant des caractéristiques temporelles différentes ; on trouve par exemple :
– des lois de commande périodiques, où le respect de contraintes de cadence et de latence conditionne les performances de la commande ;
– des lois de commande périodiques multi-cadences, où on peut affecter des caractéristiques temporelles (période, priorité...) différentes à certaines parties de l'algorithme pour en augmenter l'efficacité [40] ;
– des activités, répétitives ou non, dont la durée d'exécution dépend de l'environnement et n'est pas connue a priori (planification, traitement d'images...) ;
– des activités sporadiques, représentant par exemple les changements d'état et de configuration du système, les traitements d'exceptions ou les réactions aux pannes.

Ces diverses activités partagent un support d'exécution de capacité bornée en devant respecter des contraintes temps-réel spécifiées sous la forme d'échéances temporelles à respecter. Le respect de ces échéances peut être strict (tout dépassement est interdit) ou lâche (les commandes périodiques peuvent tolérer sans dommage des variations autour de valeurs nominales de périodes et de latences). On peut aussi graduer une contrainte stricte et définir des systèmes ¨weakly hard¨, où la contrainte dure de respect d'échéance est remplacée une distribution précisément définie de dépassements autorisés, exprimée en terme du nombre d'échéances respectées ou dépassées sur une fenêtre temporelle donnée [5]. Le modèle d'erreurs temporelles doit être complété par la définition du traitement d'exception associé devant idéalement permettre un passage graduel en fonctionnement dégradé.

3 Modèle classique d'ordonnancement temps-réel

L'exécution de ces systèmes est souvent gérée par un système d'exploitation utilisant un ordonnanceur à priorités fixes et préemption. La variété des caractéristiques temporelles et l'incertitude liée aux valeurs connues uniquement a l'exécution de ces activités font qu'il est difficile et inefficace de spécifier un ordonnancement statique (e.g. une affectation statique de périodes), utilisant des majorants et conduisant a une sous-utilisation chronique du support d'exécution. Dans ce cas les dépassements d'échéance peuvent également conduire à une avalanche de fautes temporelles et à un écroulement du système. D'autre part, dans la littérature classique concernant l'ordonnancement temps-réel, les algorithmes (par exemple la politique d'affectation des priorités) sont étudiés séparément du contexte applicatif [13]. Cette démarche peut conduire a préconiser un ordonnancement ¨optimal¨ du point de vue par exemple de l'utilisation de la ressource de calcul mais inefficace du point

de vue du contrôle de l'application qui est tout de même le point de départ [44].

Rappelons que d'une façon classique le système informatique est modélisé par un ensemble $\{\tau_1, \tau_2, ..., \tau_n\}$ de tâches, chacune d'entre elle étant caractérisée par ses paramètres temporels [25] :

- la date de première activation r_i
- la pire durée d'exécution (WCET) C_i
- le délai critique relatif (deadline) D_i
- la période d'activation P_i

Sous les hypothèses restrictives suivantes :

- toutes les tâches sont périodiques et s'exécutent sur un mono-processeur ;
- elles ont le même instant critique (tous les r_i sont égaux) ;
- il n'y a pas de relation de dépendance entre les tâches ;
- les tâches sont à échéance sur requête ($P_i = D_i$, pour tout i)

il est montré que la politique Rate Monotonic (les priorités fixes sont ordonnées dans l'ordre inverse des périodes) est optimale, i.e. tout ensemble de tâches ordonnançable de cette classe l'est par R.M. (et EDF est optimal pour des priorités dynamiques). De nombreuses extensions ont par la suite été apportées au modèle original pour, par exemple, prendre en compte les dépendances et l'exclusion mutuelle entre tâches ou encore l'existence de processus apériodiques [38].

3.1 Limites de l'approche

Connaissance des paramètres temporels Ces approches supposent que l'on connaisse au départ la valeur des paramètres temporels des tâches. Si toutes les tâches ne sont pas périodiques, le problème peut être plus ou moins bien contourné par une ¨mise en réserve¨ de temps CPU (e.g. le serveur sporadique), ce qui suppose tout de même de connaître une borne inférieure du temps séparant deux activations successives. On peut difficilement intégrer des notions d'urgence ou d'importance de ces tâches.

La détermination de la durée d'exécution des tâches est un problème difficile. Cette durée dépend évidement du langage hôte (et des optimisations éventuelles du compilateur) et de la machine cible. Pour une machine et un compilateur donné, cette durée dépend également du contexte, et est de plus en plus difficile à déterminer avec les processeurs modernes utilisant des caches et/ou des pipe-lines. Les branchements conditionnels dans le code génèrent également des durées d'exécution variables (potentiellement une durée par branche).

L'incertitude sur la durée d'exécution des tâches peut également provenir de l'algorithme lui même. Citons en particulier les processus de vision où la durée de traitement dépend de la richesse et de la complexité de la scène observée, et les algorithmes de planification de trajectoires... Les processus itératifs, où la précision du résultat dépend du nombre d'itérations effectuées, fournissent un exemple de processus où la valeur d'un critère de qualité de services peut être calculée en fonction de la précision du résultat, et donc plus ou moins explicitement en fonction de la durée de calcul effectuée. C'est en particulier le cas des contrôleurs prédictifs (model predictive control MPC), où une optimisation quadratique doit être effectuée à chaque pas [21].

Enfin, remarquons que même une connaissance de la durée d'exécution au pire cas ne garantit pas forcément l'obtention d'un ordonnancement fiable. Par exemple, [23] analyse le fonctionnement d'une installation industrielle où l'ordonnancement se déroule conformément au cahier des charges lorsque toutes les tâches sont à leur WCET et qui se bloque sur une inversion de priorité non protégée quand une tâche d'acquisition termine son exécution plus tôt que prévu...

Politique d'affectation des priorités Les politiques d'affectation des priorités étudiées dans la littérature (RM, DM, EDF...) présentent un certain caractère d'optimalité, e.g. utilisation maximale du CPU, donc du point de vue de l'informatique et en dehors de tout contexte applicatif. Or, une application de commande ordonnancée ¨en aveugle¨ selon une de ces méthodes peut exhiber un comportement médiocre, alors qu'un découpage de l'algorithme et une affectation de priorité suivant des notions d'urgence ou d'importance relatives, dépendant du contexte applicatif, peut conduire à obtenir des performances de l'application (stabilité, faible erreur de poursuite...) beaucoup plus satisfaisants. Dans [18] par exemple on montre que l'application basique de Rate Monotonic à un système de commande simple (3 pendules inversés commandés en parallèle) conduit, via la préemption, à de grandes latences de calcul entraînant l'instabilité du système. Un re-découpage du flot de données (cf. section 4) et une affectation des priorités conforme à l'urgence relative des composants permet d'obtenir un fonctionnement satisfaisant (stable pour les 3 commandes). Il est en effet important de minimiser les latences de calcul, ce qui n'est pas (ou mal) pris en compte dans les théories d'ordonnancement citées.

Robustesse et paramètres temporels Les systèmes temps-réel peuvent être classés en systèmes temps-réel ¨durs¨, où l'on interdit le moindre dépassement d'échéance, et les systèmes ¨mous¨ où des dépassements d'échéance occasionnels sont autorisés (sans plus de précision, puisque la littérature sur l'ordonnancement temps-réel ne se préoccupe pas des particularités des applications). Les systèmes de commande sont le plus cités comme exemples de systèmes temps-réel durs, sans approfondir l'argumentation. Or les systèmes de commande en boucle fermée présentent tous une certaine robustesse vis à vis des incertitudes liées

aux paramètres du processus (leur robustesse est due à une marge de phase, qui doit absorber les variations des paramètres autour de leur valeur nominale) [12]. On peut constater expérimentalement (e.g. [8]) qu'ils ont aussi une certaine robustesse vis à vis de variations des paramètres temporels de l'ordonnancement autour de leurs valeurs nominales. On peut même exhiber des systèmes de commande en boucle fermée fonctionnant très bien alors que tous les modules périodiques dépassent leurs échéances en permanence... Les effets constatés lors d'une étude de cas expérimentale rapportée dans [7], où le processus commandé est un pendule inversé et la qualité de la commande est mesurée par la variance de l'erreur d'asservissement sont les suivants (Figure 2) :

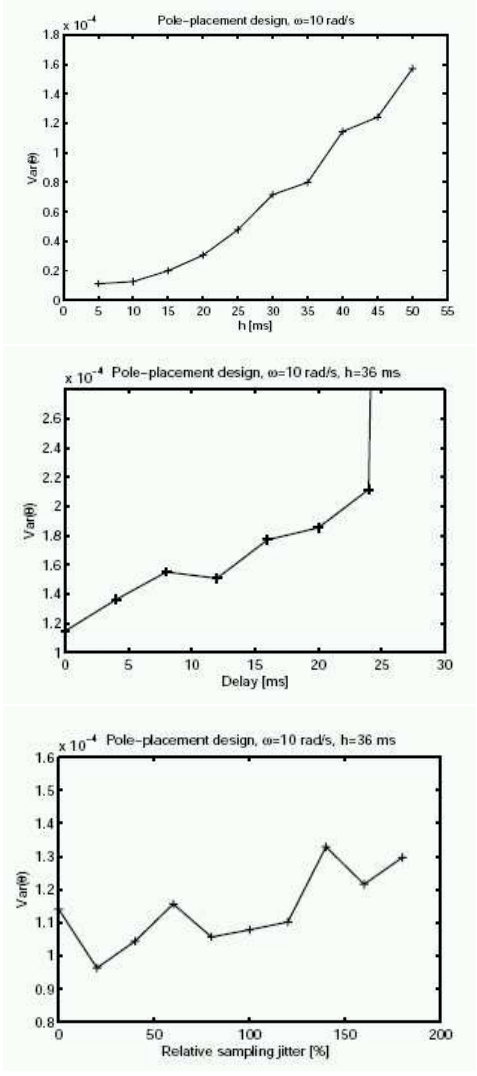

FIG. **2. Effet des variations de période, latence et gigue sur la commande d'un pendule inversé**

- La qualité de commande croît quand la période d'échantillonnage diminue. Cette qualité de commande augmente même si l'on ne profite pas de l'accroissement de la fréquence d'échantillonnage pour augmenter les gains du régulateur [22], [40]. La valeur maximum possible pour la période peut être évaluée grâce à la théorie des systèmes linéaires échantillonnés (théorème de Shannon) ;
- la qualité décroît quand la latence augmente. La détérioration due à la latence croît plus rapidement que celle due à l'augmentation de la période, avec une limite conduisant rapidement à l'instabilité du système (dans l'exemple présenté figure 2 l'algorithme de commande ne tient pas compte du retard ; cette détérioration pourrait être amoindrie avec une commande tenant compte du retard, mais plus compliquée et donc augmentant la période. . .) ;
- la gigue sur l'émission des sorties détériore la qualité, mais l'asservissement reste stable même pour des valeurs de gigue importantes. L'effet de la gigue sur les mesures dépend de l'utilisation qui en est faite. Si l'on mesure directement l´état du système pour l'utiliser dans le calcul de la commande, cet effet n'est pas plus important que celui dû à la gigue de sortie. Cet effet est par contre très sensible si l'on utilise ces mesures dans un processus d'identification ou de reconstruction d'état (imaginer la sortie d'un simple pseudo-dérivateur reconstituant la vitesse à partir des positions par $\hat{q_n} = \frac{(q_n - q_{n-1})}{dt}$ si dt n'est pas constant. . .).

4 Conception conjointe commande et ordonnancement

Une première approche consiste à calculer, hors-ligne, une affectation des paramètres d'ordonnancement maximisant (idéalement) la performance de commande du système sous contrainte d'ordonnançabilité des tâches. Le point de départ est bien entendu d'obtenir un modèle de performance fonction des paramètres d'implémentation.

Par exemple, la méthode décrite dans [36] et [37] utilise une fonction de coût entre un critère de performance quadratique et la période d'échantillonnage du contrôleur. Si cette fonction est convexe l'algorithme proposé choisi, parmi toutes les configurations de tâches ordonnançables par la politique choisie (RMA ou EDF) celle maximisant la performance de commande.

Une autre possibilité consiste à formuler le problème d'implémentation de commande sous forme d'un critère de qualité de service rendant compte, par exemple, de la relation performance/période de chaque contrôleur dans le système. Ce modèle peut être utilisé pour configurer le contrôleur d'admission gérant la charge du système [1] ou encore pour négocier en ligne périodes et priorités comme proposé

par [33].

Lorsque plusieurs tâches de commande s'exécutent sur une ressource partagée, la préemption résultant de la concurrence induit des latences de calcul dépendant non seulement de la durée de calcul des fonctions mais aussi de l'entrelacement de leurs instants d'exécution. La méthode décrite dans [31] et [32] utilise des modèles de performance de commande basé sur la théorie des systèmes linéaires, où la fonction de coût décrit la performance (e.g. temps de réponse) du système en fonction des deux paramètres période et retard. Une heuristique itérative d'optimisation ajuste alors les paramètres d'implantation de façon à maximiser la performance globale tout en respectant la faisabilité de l'implémentation. La méthode a été validée hors-ligne mais paraît trop complexe pour être utilisée en temps-réel.

Les approches précédentes supposent que le découpage des algorithmes de commande en tâches est prédéfini : en fait toutes les composantes d'un logiciel de commande n'ont pas la même importance relative si l'on se réfère à leur impact sur une fonction de coût performance/paramètres temporels. Un découpage de l'algorithme de commande conscient de l'urgence et de l'importance relative des composantes permet en particulier de minimiser la latence de calcul sur certains chemins critiques.

Ainsi, il est possible de partitionner le programme du contrôleur d'un système linéaire de la façon suivante [4] :

```
while(1){
Wait-Clock
A/D-Conversion
Calculate-Output /*calcul commande u_k*/
D/A-Conversion
Update-State /*mise à jour modèle x̂_k*/
}
```

où la latence mesure/commande est minimisée en envoyant la commande dès que calculée, la mise à jour du modèle pouvant se faire plus tard, n'importe quand avant la période suivante. Cette méthode est par exemple illustrée dans [18], où ce découpage appliqué à la commande de pendules inversés concurrents apporte une amélioration spectaculaire des performances de commande, sans coût supplémentaire en charge de calcul. Les systèmes non-linéaires peuvent également, et sans doute encore plus, bénéficier d'un découpage adéquat et d'une implantation multi-cadences des processus de commande [39], par exemple en séparant en tâches temps-réel ordonnancées séparément une boucle rapide de stabilisation et une boucle lente de perception-navigation. Un autre exemple concernant la commande de robot est donné dans la section 6.5.

Enfin, du côté informatique des solutions ont été proposées pour prendre en compte les contraintes de la commande dans l'implémentation des contrôleurs. Citons par exemple le modèle des tâches élastiques [6], où l' ¨élasticité¨ des tâches modélise leur sensibilité aux variations de performance (mesure d'un QdS fonction de la période) et permet à l'ensemble de tâches de ¨comprimer¨ harmonieusement leurs périodes pour s'adapter aux surcharges. Citons également le Control-Server [9], où une fraction de la puissance de calcul disponible est réservée à chaque contrôleur : du point de vue de l'exécution sur la ressource partagée chaque contrôleur est isolé, un contrôleur en surcharge temporaire n'a ainsi pas d'impact sur l'exécution des autres. À l'intérieur de chaque segment réservé les tâches sont organisées de façon à minimiser latence et gigue.

5 Ordonnancement régulé (Feedback-scheduling)

Un certain nombre des aspects temps-réel présents dans les systèmes de contrôle/commande ne sont pas convenablement traités par les méthodes classiques d'ordonnancement à priorités fixes. Essentiellement, ce sont :

- l'analyse d'ordonnancement suppose que l'on connaisse les WCET des différentes tâches, ce qui est généralement difficile à obtenir (et même de plus en plus difficile avec les processeurs modernes à multiples niveaux de cache et de pipe-line). De nombreux algorithmes, par exemple en traitement d'image, ont une durée d'exécution dépendant des entrées (nombre de segments dans la scène), inconnue à priori. Tout ordonnancer au temps d'exécution maximum prévisible conduit à une sous-utilisation chronique et importante des ressources de calcul ;
- la charge de calcul peut varier fortement au cours du temps : là encore le traitement d'images et la commande référencée capteurs en robotique en donnent des exemples (traitement dépendant du nombre de capteurs actifs à un instant donné). On peut aussi trouver des exemples dans la gestion de serveurs multimédias [1] ;
- la relation performances/ordonnancement d'une loi de commande est mal connue, surtout quantitativement, et l'affectation des paramètres temporels reste basée sur des résultats non exhaustifs et non généralisables d'expériences et de simulations [40].

L'ordonnancement à priorités dynamiques peut apporter une certaine souplesse en réaffectant en ligne la priorité des tâches en fonction, par exemple, de l'ordre des échéances comme dans l'algorithme EDF [13]. Le coté ¨optimal¨ de l'ordonnancement produit se paye cependant par l'instabilité du système en cas de surcharge (effet domino) et, semble-t-il, par un overhead d'exécution important. Surtout il ne semble pas exister (à notre connaissance) d'O.S. diffusé utilisant un tel ordonnanceur à priorités dynamiques. Enfin, comme déjà mentionné, les décisions d'ordonnancement prises sur un critère purement informatique (maximiser la charge CPU en respectant le maximum d'échéances)

à la fréquence du scheduler ne tiennent pas compte (ou indirectement) des spécificités de l'application. Le fonctionnement d'un tel ordonnanceur peut être amélioré par l'addition d'un processus de supervision, comme le régisseur d'ordonnancement proposé par [16].

Une solution alternative consiste à adapter en temps-réel l'ordonnancement grâce à un régulateur en boucle fermée, par exemple en ajustant la période d'activation de tâches en fonction d'une mesure de qualité de commande (QoC) reflétant les objectifs de performance du système.

L'idée nouvelle, principalement étudiée au laboratoire d'Automatique de l'université de Lund[1], au département d'informatique de l'université de Virginie[2] et aussi récemment en collaboration entre l'INRIA et le LAG [3] consiste à ajuster en boucle fermée les paramètres d'ordonnancement du système en fonction de mesures faites sur celui-ci à l'aide d'un algorithme de commande : on peut alors parler d'*ordonnancement régulé* (ou *feedback scheduling*).

Le principe général de l'architecture est décrit par la figure 3. On trouve bien au niveau de l'ordonnanceur la chaîne mesure/régulateur/actionneur d'une boucle de commande. On y trouve plusieurs processus particuliers :

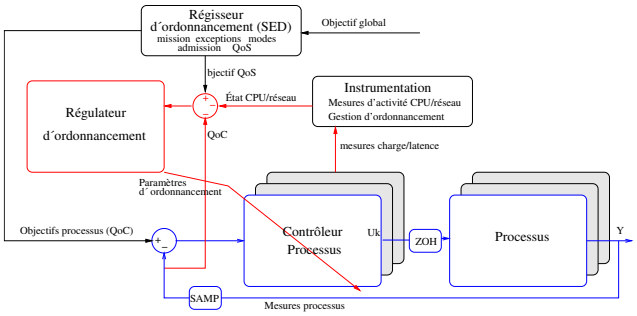

FIG. **3. Ordonnanceur régulé**

- Le *régulateur d'ordonnancement* calcule les paramètres d'ordonnancement (e.g. période, niveau de QoS) en fonction d'un signal d'erreur mesuré sur le processus commandé (e.g. erreur de poursuite) et/ou sur le contrôleur (e.g. charge CPU) ;
- Le *régisseur d'ordonnancement* calcule l'objectif de fonctionnement désiré du système informatique (e.g. consigne de charge CPU), gère l'admission de nouvelles requêtes, traite les exceptions et le passage en mode dégradé...
- Le système doit être *instrumenté* pour permettre d'implémenter ces deux nouvelles tâches ; il s'agit de mesurer l'état et l'activité du système informatique et de manipuler les paramètres d'ordonnancement des contrôleurs de processus. Il peut s'agir d'une couche intergicielle, entre l'O.S. temps-réel et l'application de contrôle/commande.

[1]http ://www.control.lth.se/ anton/artes/
[2]http ://www.cs.virginia.edu/ stankovic/rts.html
[3]http ://www.inrialpes.fr/pop-art/people/simon/c3o/c3o.html

5.1 Capteurs et mesures

Le signal d'erreur est élaboré à partir de mesures prises sur le processus commandé (estimation de QoC) et sur le système informatique. L'état du processus contrôlé est obtenu classiquement à partir de capteurs et d'algorithmes de reconstruction/filtrage.

L'état de la ressource d'exécution peut être obtenu au travers de différentes mesures.

5.1.1 Charge CPU

Une contrainte incontournable dans l'exécution des contrôleurs est la saturation du CPU, se traduisant par des surcharges et des dépassements d'échéances. La charge de calcul induite par les différents modules de programme s'exécutant sur la ressource partagée doit être mesurée ou estimée, cette mesure pouvant être ensuite utilisée par le régulateur d'ordonnancement comme contrainte, comme variable de commande ou encore comme perturbation.

Il faut pour commencer obtenir une estimation $\hat{c}$ de la durée de calcul de chaque tâche temps-réel. Ce n'est possible par mesure directe que si l'OS est instrumenté pour cela (c'est par exemple le cas de RTAI). On peut aussi faire une estimation par le biais du temps de réponse (possible par l'API Posix), qui ne tient pas compte des instants de préemption et donc surestime la charge totale : la synthèse d'un estimateur robuste, simple et non biaisé de la charge CPU à partir des temps de réponse reste à faire...

À partir de l'estimation $\overline{c_i}_{kh_s}$ de la durée de calcul moyenne de la tâche i pendant la kième période h_s de mesure, l'estimation filtrée de la charge induite par cette tâche pendant une période h_s est :

$$\hat{U}_{kh_s} = \lambda\, \hat{U}_{(k-1)h_s} + (1-\lambda)\, \frac{\overline{c_i}_{kh_s}}{h_{((k-1)h_s)}}$$

où λ est facteur d'oubli. L'utilisation des fréquences d'échantillonnage $f_i(kh_s) = 1/u_i(kh_s)$ permet d'obtenir un modèle linéaire de la charge induite par les n tâches de commande périodiques :

$$\hat{U}(kh_S) = \frac{(1-\lambda)}{z-\lambda} \sum_{i=1}^{n} \overline{c}_i(kh_S) f_i(kh_S)$$

.

Un paramètre important est la valeur de la période d'échantillonnage h_s des mesures et du régulateur d'ordonnancement. Cette valeur doit être assez grande (plus grande que tous les h_i des tâches concernées) pour que la mesure obtenue ne soit pas trop bruitée [27]. Ce modèle (dit ¨fluide¨) n'a de sens que si l'on regarde le système ¨d'assez

loin¨, et interdit donc de réagir rapidement à des perturbations soudaines. Le régulateur obtenu devra être complété par un mécanisme de traitement des surcharges transitoires occasionnés par les changements brusques du point de fonctionnement.

5.1.2 Autres mesures possibles

D'autres mesures possibles concernant l'état de la ressource d'exécution sont :

- les dépassements d'échéance sont des événements faciles à détecter ; leur nombre par unité de temps (miss ratio dans [28]) ne donne une mesure non nulle qu'en cas de surcharge du CPU. Ils peuvent cependant être utilisés en complément de l'estimation de charge (cf par exemple 6.1) ;
- la laxité des tâches (temps restant entre la fin d'exécution d'une instance et l'instant d'activation suivant) peut aussi être envisagée comme indicateur de charge du système ;
- dans le cas d'un système distribué, on devra aussi mesurer ou estimer la charge du réseau, les retards induits, les pertes de messages...

5.2 Actionneurs et commandes

Les variables d'action disponibles sont :

- Période des tâches : la charge CPU induite par n tâches de durée c_i et de période h_i est $U = \sum_{i=1}^{n} \frac{C_i}{h_i}$: on voit immédiatement que les périodes sont des actionneurs efficaces pour agir sur la charge globale de calcul. Le système doit fournir des outils ou une API permettant de modifier en ligne les périodes affectées aux tâches ;
- Priorités des tâches : l'ordre des priorités n'affecte pas la charge de calcul mais l'entrelacement des calculs, et donc les latences mesure/commande. Les priorités doivent aussi refléter l'urgence et l'importance relative des composants sur la performance du contrôleur ;
- L'utilisation de variantes {coût d'exécution, performance} d'une même fonctionnalité peut aussi être utilisé comme actionneur sur la charge de calcul, à l'échelle de temps des changements de mode de marche. Ces variantes peuvent être associée à des valeurs de contribution au QoS de l'application ;
- Le régisseur du système est un système à événements discrets gérant globalement pour l'application et la ressource d'exécution partagée l'admission, le rejet ou l'ajournement de nouvelles requêtes d'activation.

6 Exemples de régulateurs d'ordonnancement

Nous allons maintenant donner quelques exemples d'ordonnancement régulé : suivant la complexité de mise en oeuvre et les objectifs de commande choisis les algorithmes de commande sont adaptés à partir de ceux trouvés dans la boîte à outils de l'Automaticien, de la simple commande PID mono-variable à la commande optimale. Devant la difficulté à modéliser précisément le comportement du système, où les incertitudes viennent aussi bien des processus commandés que de la ressource d'exécution, la notion de robustesse est essentielle.

6.1 Commande de serveur par P.I.D.

La formule de base d'un régulateur P.I.D. (Proportionnel-Intégral-Dérivé) est en temps continu [4] :

$$U = K_p.e + K_v.\frac{de}{dt} + K_i.\int_0^t e(\tau).d\tau$$

où U est la commande appliquée au processus et e le signal d'erreur entre la consigne y_d et la sortie mesurée y (figure 4).

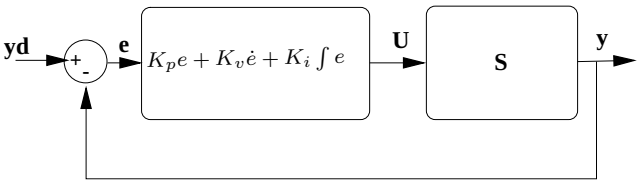

FIG. **4. Boucle de commande**

Il s'agit d'un type de régulateur très répandu car applicable à un grand nombre de processus mono-variables. Il en existe des versions en temps continu (analogiques) et numériques.

Si la réponse en boucle ouverte du processus correspond approximativement à la figure 5, on peut le caractériser par les trois paramètres (signature) L, T et R, ce qui représente un faible effort de modélisation. Un PID est en particulier susceptible de donner des résultats satisfaisants si le rapport T/L est grand (¨retard¨ faible devant la ¨constante de temps¨). Suivant les performances recherchées, la nature du processus et les contraintes de complexité, on peut se contenter d'un régulateur réduit, par exemple P, PI ou PD.

Un technicien un peu expérimenté peut régler un tel régulateur sur le terrain en observant la réponse du système bouclé : le gain proportionnel K_p diminue le temps de réponse, le gain dérivé K_v augmente l'amortissement et gomme les oscillations, enfin le gain intégral K_i annule l'erreur statique.

Il existe d'autre part un certain nombre de méthodes de détermination hors ligne des gains K_p, K_v et K_i : synthèse fréquentielle utilisant la signature, méthode semi-empirique de Ziegler et Nichols, modelage du transfert de boucle [15]... Il est également possible de privilégier la robustesse aux erreurs et variations des paramètres du processus plutôt

que d'augmenter les performances dynamiques en utilisant une méthode de réglage des gains spécifique (PID robuste [14]).

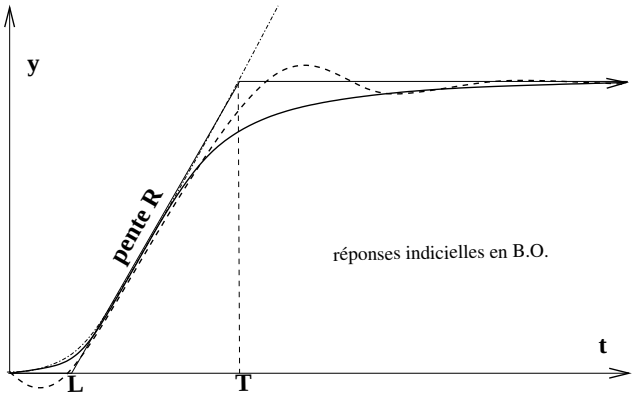

FIG. **5. Caractéristiques en boucle ouverte**

La simplicité conceptuelle du régulateur PID a conduit certains chercheurs a l'utiliser pour commander et réguler le fonctionnement d'un système informatique de type serveur. Cette démarche est en effet bien adaptée à la commande de système à Qualité de Service (QoS), où les contraintes temporelles sont relatives, les dépassements occasionnels autorisés (mais pénalisés) et où la charge (requêtes des clients) est très variable et peu prédictible.

On trouve ainsi le principe de commande en boucle fermée appliqué à l'administration de serveurs web ([2], [26]) ou de serveurs e-mail ([29]). Les principes généraux, des modèles et des études de cas de commande en boucle fermée de systèmes informatiques sont décrits dans [20].

La figure 6 représente le type de d'architecture étudiée dans [27] et [28] détaillée dans cette section.

6.1.1 Modèle du serveur

Le serveur doit exécuter les tâches requises en ligne par les utilisateurs. À chaque tâche est associée une ou plusieurs valeur(s) de QoS, contribuant à la valeur globale du service lorsque la tâche est terminée avec succès, soit avant son échéance. Le but du contrôleur est de maximiser la qualité du service rendu par le serveur, la contrainte étant la saturation du CPU. Dans sa forme la plus générale, le système dispose de plusieurs actionneurs :

- l'ordonnanceur temps-réel est supposé préemptif ; il peut être à priorités fixes et utiliser les politiques Rate Monotonic ou Deadline Monotonic, ou encore à priorités dynamiques et appliquer EDF ;
- le contrôleur de QoS détermine dynamiquement le niveau de qualité auquel va être exécutée chaque tâche admise en fonction de la disponibilité du CPU ;

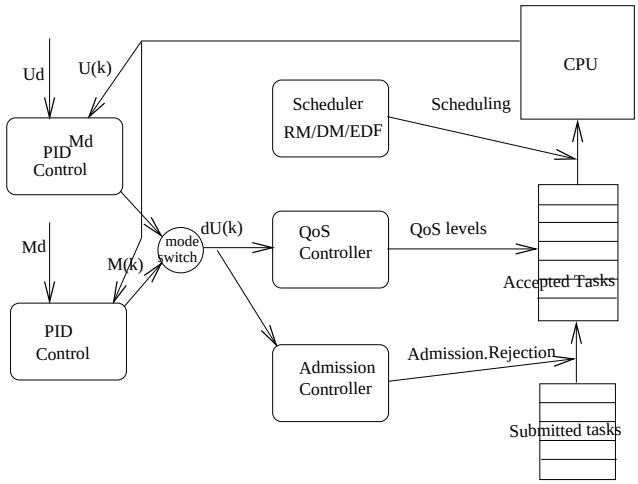

FIG. **6. Contrôle en boucle fermée d'un serveur informatique**

- le contrôleur d'admission admet ou rejette les requêtes incidentes.

Le modèle d'une tâche (pouvant être périodique ou apériodique) T_i est caractérisé par les paramètres suivants :

- N nombre de niveaux de QoS, $N \geqslant 2$;

à chaque niveau j de QoS sont associés :

- $D_i[j]$ échéance relative ;
- $EE_i[j]$ temps d'exécution estimé ;
- $AE_i[j]$ temps d'exécution réel, variable d'une exécution à l'autre et inconnu de l'ordonnanceur ;
- $V_i[j]$ valeur de QoS lorsque la tâche est exécutée avec succès. $V_i[0] \leqslant 0$ est la pénalité payée au QoS global en cas d'insuccès (échéance dépassée ou tâche rejetée par le contrôleur d'admission).

pour les tâches périodiques

- $P_i[j]$ période d'activation
- $B_i[j]$ charge CPU estimée $B_i[j] = EE_i[j]/P_i[j]$
- $A_i[j]$ charge CPU réelle $A_i[j] = AE_i[j]/P_i[j]$

pour les tâches apériodiques

- $EI_i[j]$ intervalle estimé entre deux arrivées de la tâche
- $AI_i[j]$ intervalle moyen (inconnu de l'ordonnanceur)
- $B_i[j]$ charge CPU estimée $B_i[j] = EE_i[j]/EI[j]$
- $A_i[j]$ charge CPU réelle $A_i[j] = AE_i[j]/AI[j]$

6.1.2 Modèle de commande

Les variables mesurées et contrôlées par l'ordonnanceur sont :

- le taux d'échéances dépassées (miss ratio) $M(k)$, défini comme étant le rapport entre le nombre de dépassement d'échéances sur le nombre de tâches exécutées (avec succès ou abandonnées) sur la période $[(k-1)W, kW]$ du contrôleur ;

– l'utilisation du CPU $U(k)$ sur la fenêtre temporelle $[(k-1)W, kW]$, on remarque qu'une saturation du CPU induit automatiquement des dépassements d'échéance ;
– la valeur de qualité totale $V(k)$ sur cette même fenêtre temporelle pourrait être utilisée, en fait cette valeur agit indirectement via l'actionneur de QoS.

La variable d'action est la charge totale estimée $B(k) = \sum_i B_i[l_i(k)]$, sommant la contribution de chaque tâche T_i affectée du niveau de QoS l_i à la kième période d'échantillonage. Elle agit en manipulant les valeurs de QoS l_i affectées aux tâches admises, et également en régulant l'admission de nouvelles requêtes. Sa variation est donnée par $B(k+1) = B(k) + D_B(k)$.

La référence de performance est donnée par des consignes de charge CPU U_S et/ou de taux d'échéances dépassées, par exemple $U_S = 90\%$ *et* $M_S = 0$. L'objectif de commande est donc constitué de deux sous-objectifs complémentaires :

– La commande de charge CPU n'est effective que quand le CPU n'est pas saturé ($U_{real} \leqslant 100\%$). En cas de surcharge une augmentation des requêtes d'exécution ne se traduit plus par une augmentation du nombre de requêtes servies avec succès, mais uniquement du taux d'échecs.
– Quand le CPU n'est pas saturé (et si l'ordonnancement est bien conçu) il ne devrait pas y avoir (ou peu) de dépassements d'échéances. La mesure du taux de dépassement n'est normalement effective qu'en cas de surcharge, lorsque la charge réelle devient supérieure à une charge maximum compatible avec la politique d'ordonnancement (par ex. 1 avec EDF).

Une difficulté prévisible vient de ce que le point de fonctionnement choisi (charge CPU élevée) est proche de la saturation de l'actionneur où le système change radicalement de caractère (en particulier du point de vue de sa commandabilité). On peut cependant montrer que l'utilisation d'un au moins de ces deux actionneurs est toujours possible à un instant donné. Il est donc possible de commander le serveur au moyen de deux contrôleurs mono-variables alternants (fig 6). (On peut aussi utiliser une formulation multi-variables comme dans [17] pour spécifier des objectifs de commande multiples).

On appelle G_A la borne maximum du rapport d'utilisation charge réelle/charge requise ($G_A = max(A(k)/B(k))$) et G_M la borne maximum du rapport entre incrément de dépassements et incrément de charge ($G_M = max(dM(k)/dA(k))$).

Dans les plages non saturées, les modèles linéarisés d'utilisation et de dépassement sont :

$$\begin{cases} U(k) = U(k-1) + G_A.D_B(k-1) \text{ si } A(k) \leqslant 1 \\ \text{ sinon } U(k) = 1 \\ M(k) = M(k-1) + G_M.G_A.D_b(k-1) \text{ si } A(k) > A_{th} \\ \text{ sinon } M(k) = 0 \end{cases}$$

En appelant $X(z)$ la transformée en z de la variable $x(k)$ où z^{-1} est l'opérateur retard ($x_{k-1} = z^{-1}x_k$), on obtient un modèle de comportement du CPU sous forme de fonctions de transfert (en fait de simples filtres passe-bas) :

$$\begin{cases} P_U(z) = U(z)/D_B(z) = \frac{G_A}{(z-1)} \text{ si } A(k) \leqslant 1 \\ P_M(z) = M(z)/D_B(z) = \frac{G_A.G_M}{(z-1)} \text{ si } A(k) > A_{th} \end{cases}$$

On a ainsi obtenu un modèle linéaire échantillonné du comportement du CPU vis à vis de la charge demandée et du respect des échéances. Cette abstraction n'a de sens que si l'on observe de loin le système (qui est par nature un S.E.D.), avec une période d'échantillonnage suffisamment grande pour moyenner et filtrer efficacement l'activité sporadique du système. En pratique la période du régulateur d'ordonnancement doit être très supérieure à toutes les périodes des tâches servies par le système, ce qui interdit *a priori* a l'ordonnanceur régulé de réagir rapidement à une perturbation soudaine. [20] décrit d'autres modèles utilisés pour la commande de systèmes informatiques, dits ¨modèles fluides¨, où par analogie le comportement des files d'attente est assimilé à des réservoirs et celui des contrôleurs à des vannes, les flots d'opérations réalisées par le système étant naturellement modélisés par des débits.

6.1.3 Contrôleur

Le contrôleur doit assurer la stabilité du système, une erreur statique nulle, l'insensibilité aux variations de charge et enfin un temps de réponse et un dépassement de consigne acceptables.

Le régulateur est mono-variable, la commande calculée est une variation de charge D_B ensuite traduite en niveaux de qualité l_i par l'actionneur de QoS (par exemple de type ¨highest-value-density-first¨ [42]).

$$\begin{cases} D_{B_U}(k) = K_{pu}.E_U(k) \text{ où } E_U(k) = U_s - U(k) \text{ si } U \leqslant 1 \\ D_{B_M}(k) = K_{pm}.E_M(k) \text{ où } E_M(k) = M_s - M(k) \text{ si } M > 0 \\ D_B(k) = min(D_{B_U}(k), D_{B_M}(k)) \end{cases}$$

Notons qu'il s'agit d'un simple régulateur proportionnel. Une action intégrale est inutile car présente dans l'actionneur de QoS (ce qui doit assurer la nullité de l'erreur statique). Enfin, compte tenu de l'activité irrégulière du système une action dérivée ne pourrait qu'apporter un niveau de bruit inacceptable.

On peut alors en déduire la fonction de transfert du système bouclé, qui est de la forme :

$$\begin{cases} H_s(z) = \frac{K_p G}{z-(1-K_p G)} \\ Y(z) = H_s(z)\frac{z}{z-1}S \quad \text{(réponse à l'échelon)} \\ G = G_A \text{ ou } G_A G_M \quad \text{(suivant le mode)} \end{cases}$$

Cette structure très simple de régulation ne permet de disposer que d'un paramètre de réglage : le choix d'une valeur particulière de K_p permet de placer (théoriquement) le pôle $p_0 = 1 - K_p G$ du transfert en boucle fermée. L'analyse de ce modèle par les méthodes classiques permet alors de valider simplement les propriétés recherchées de stabilité, d'insensibilité et de robustesse. En particulier, le fait d'avoir calculé le gain K_p en fonction des pires cas largement évalués des facteurs d'utilisation $G_A > G_a(k)$ et de perte $G_M > G_m(K)$ garantit la stabilité du système pour les valeurs réelles des paramètres (robustesse en stabilité). La simplicité du contrôleur se paye par contre par une sensibilité des performances aux variations de paramètres. Enfin, le régulateur d'ordonnancement ne s'occupe pas des priorités, qu'il faut affecter aux tâches selon une politique adéquate.

Les figures 7 et 8 montrent quelques résultats de simulation pour différents profils de charge (détaillés dans [28]) et de politique d'ordonnancement. Le gain du régulateur est choisi pour avoir un temps de réponse à 2% de 4.5 secs (compte tenu du modèle de charge), la période de régulation est choisie empiriquement pour obtenir un compromis acceptable entre stabilité et réactivité.

- DM/PA : tâches périodiques et apériodique, priorités statiques Deadline Monotonic
- EDF/P : tâches périodiques uniquement, priorités dynamiques EDF
- $K_{pu} = 0.185$, $K_{pm} = \begin{cases} 0.414 & \text{si DM/PA} \\ 0.148 & \text{si EDF/P} \end{cases}$
- période du régulateur : 0.5 sec

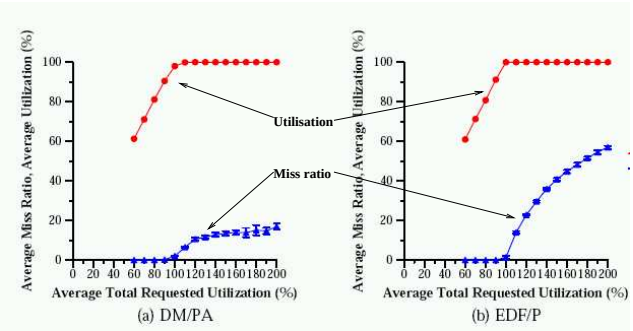

FIG. **7. Réponse en régime permanent**

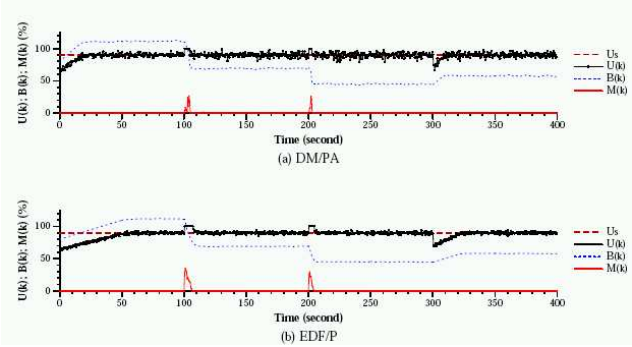

FIG. **8. Réponse transitoire à une surcharge**

On peut observer, en particulier sur la figure 7 donnant le régime permanent (en utilisation et en pertes) fonction de la charge de consigne, l'effet de la politique d'ordonnancement sous-jacente : l'utilisation de EDF permet de reculer jusqu'à $U_d = 1$ le début des dépassements d'échéance, mais en cas de surcharge permanente le comportement du système se dégrade plus rapidement qu'avec la politique à priorité statique DM. La réaction du système à des surcharges transitoires (8) montre le retour des variables contrôlées vers leur valeur de consigne avec une dynamique proche de celle spécifiée.

6.2 Commande optimale

L'approche par régulation de type P.I.D., décrite au paragraphe précédent, a le mérite de la simplicité dans sa conception, son analyse et dans le réglage des gains de commande. Cette simplicité en donne aussi la limite, le nombre très limité de paramètres réglables interdisant, par exemple, de découpler a volonté les transferts du système de commande. Le problème traité était également simplifié puisque le seul processus commandé en boucle fermée était le serveur, les processus contrôlés (e.g. cadencement d'un flot mpeg) étant eux en boucle ouverte.

Revenons au problème initial de commande de processus : le problème consiste à *optimiser* une performance de commande *sous contrainte* d'utilisation bornée de la ressource d'exécution. Ce problème peut être résolu analytiquement dans le cas suivant [19], [8] :

On doit exécuter sur une ressource de calcul partagée n tâches temps-réel, chacune étant le contrôleur d'un système linéaire stochastique ; ces contrôleurs ont pour période h_i et une durée d'exécution C_i. Un critère de performance, dépendant de la fréquence d'échantillonnage, $J_i(h_i)$ est associé à chaque contrôleur i. Il s'agit de maximiser une fonction de coût globale pour l'ensemble des contrôleurs sous contrainte de respect d'un niveau de charge spécifié U_d du calculateur, soit :

$\min_n J = \sum_{i=1}^{n} J_i(h_i)$ sous la contrainte $\sum_{i=1}^{n} C_i/h_i \leqslant U_d$

Le problème a pu être résolu en utilisant pour chacun des contrôleurs une fonction de coût de la forme $J(h) = \frac{1}{h}\int_0^h \begin{bmatrix} x^T(t) & u^T(t) \end{bmatrix} Q \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} dt$

On observe sur la figure 9 de telles fonctions de coût en fonction de la période h, correspondant à la commande d'un pendule (inversé à gauche et stable à droite). On constate que cette fonction peut ne pas être convexe, les pics correspondants dans ce cas aux résonances en boucle ouverte du processus commandé (mais ceci est dû au choix particulier de J).

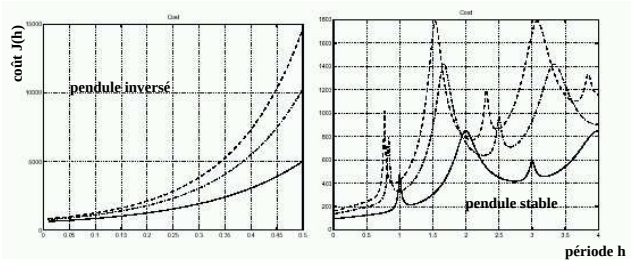

FIG. **9. Exemples de fonctions de coût**

Le choix de cette fonction de coût particulière permet d'obtenir une formulation analytique du contrôleur en boucle fermée réalisant l'optimisation, et pouvant être réalisé sous forme d'un retour d'état. Cette implémentation, nécessitant la résolution en temps-réel d'équations de Lyapunov et de Ricatti, est cependant beaucoup trop coûteuse, en comparaison du coût des contrôleurs de processus eux mêmes, pour être réalisée.

Il existe heureusement des solutions approximatives beaucoup plus simples dans le cas où les fonctions de coût peuvent être approchées par des fonctions quadratiques $J_i(h) = \alpha_i + \beta_i h^2$ ou linéaires $J_i(h) = \alpha_i + \gamma_i h$. Le calcul donnant la valeur optimale des h_i devient particulièrement simple si les fonctions de coût des différents contrôleurs sont *toutes* linéaires ou *toutes* quadratiques [10].

Dans ce cas, le calcul des périodes est obtenu comme suit :

- les valeurs initiales des fréquences de commande $f_i = 1/h_i$ sont choisis en proportion de $(\beta_i/C_i)^{1/3}$ (coûts quadratiques) ou de $(\gamma_i/C_i)^{1/2}$ (coûts linéaires) ;
- ces valeurs correspondent à une charge nominale : $\hat{U_0} = \sum_{i=1}^{n} \frac{\hat{C_i}}{h_{0i}}$
- estimation et filtrage des durées des contrôleurs (filtre passe-bas, λ est un facteur d'oubli : $\hat{C_i}(k) = \lambda \hat{C_i}(k-1) + (1-\lambda)c_i$
- les période des tâches pour d'autres consignes de charge sont données par une simple mise à l'échelle : $h_i = h_{0i}\frac{U_{sp}}{\hat{U_0}}$
- suivant la complexité des contrôleurs, les gains de commande dépendants des périodes h_i sont soit tabulés, soit recalculés en ligne ;
- les consignes de charge totale U_{sp} sont élaborées par un composant de supervision appelé ¨feedforward¨, et jouant en particulier le rôle d'un contrôleur d'admission.

La figure 10 présente quelques résultats de simulation utilisant TrueTime, une boite à outils pour Matlab/Simulink permettant de simuler un modèle de contrôleur incluant les effets de l'implémentation (O.S. temps réel et bus de terrain [11]).

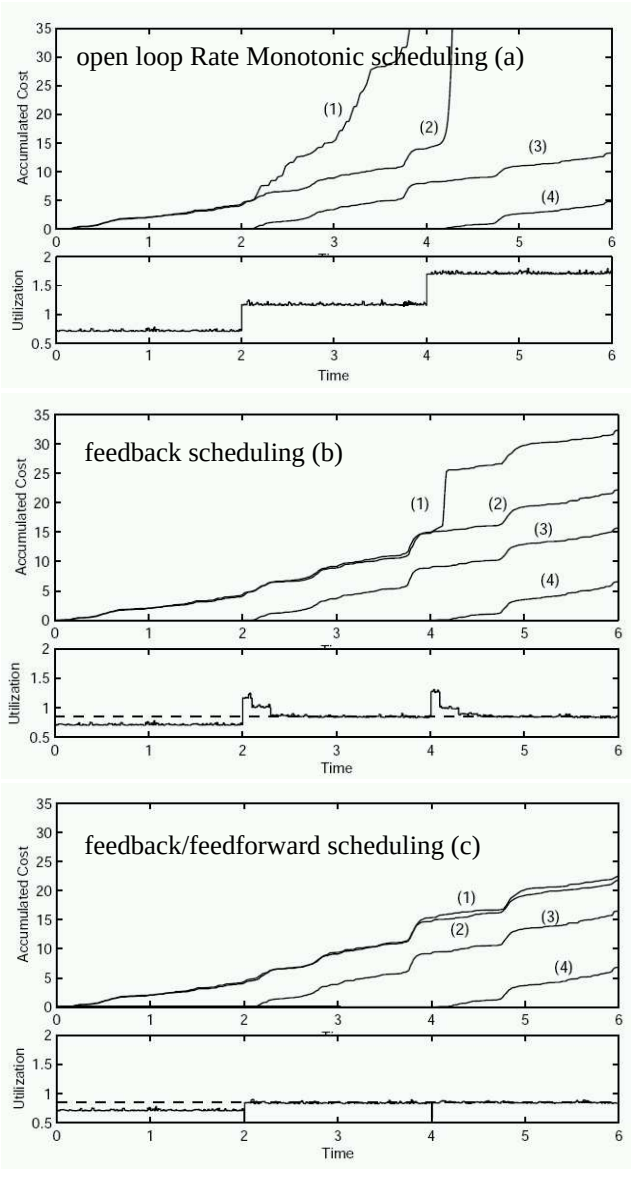

FIG. **10. Simulation TrueTime (commande de pendules)**

Dans cette simulation, on exécute 4 tâches de stabilisation de pendules inversés $T_i, i = 1...4$, dont l'ordre croissant des priorités fixes est $T_1 \prec T_2 \prec T_3 \prec T_4$. Le critère de performance choisi est le coût quadratique classique $J_i = \int_0^{Tsim}(y_i^2(t) + u_i^2(t))dt$. T_1 et T_2 s'exécutent au démarrage du système, T_3 est admise à t = 2 secs et T_4 à t = 4 secs.

Sans adaptation (10a) les contrôleurs s'exécutent à leur fréquence nominale, le calculateur devient surchargé et les tâches les moins prioritaires T_1 et T_2 sont si préemptées qu'elles ne peuvent plus stabiliser leur pendule (le critère devient infini).

L'ordonnancement régulé (10b), en recalculant en ligne les périodes, évite la surcharge du processeur et permet de maintenir tous les systèmes stables (au prix d'une baisse de performance fonction des priorités). L'adjonction d'un contrôleur d'admission (feedforward 10c) permet d'anticiper l'admission de nouvelles tâches et d'améliorer le comportement transitoire de l'ordonnanceur.

6.3 Commande robuste au retard

Les latences dans une boucle de commande sont très perturbantes : elles proviennent des durées de calcul des fonctions de commande, mais aussi de la préemption par des tâches de priorité supérieure au contrôleur en cours d'exécution ; dans le cas d'un système distribué, le réseau est également une source de retards importants (pouvant être supérieurs à la période du contrôleur).

Ces retards provenant de l'implémentation sont difficiles à modéliser précisément, ce qui rend coûteuses ou peu efficaces les méthodes de compensation. Il est plus facile d'estimer une borne supérieure du retard subi : la méthode présentée dans [35] exploite un résultat théorique récent [24] pour synthétiser une commande de processus continu robuste à un retard borné et un ordonnancement régulé par commande H_∞ pour la boucle externe. L'exemple choisi concerne l'exécution de deux tâches de commande concurrentes partageant l'utilisation d'un CPU.

Commande de l'ordonnanceur

- On utilise un modèle normalisé de l'activité CPU : $H(z^{-1}) = \frac{(1-\lambda)z^{-1}}{1-\lambda z^{-1}}$ obtenu par normalisation et transformée en z du modèle de charge linéaire établi section 5.1.1. La valeur de λ (comprise entre 0 et 1) permet de régler la rapidité d'estimation et le niveau de bruit. Les fréquences normalisées calculées par le régulateur via le modèle linéaire sont pondérées par les estimations de durée, écrêtées pour éviter sur et sous-échantillonnage, puis converties en périodes qui sont les actionneurs manipulés par l'OSTR (figure 11).

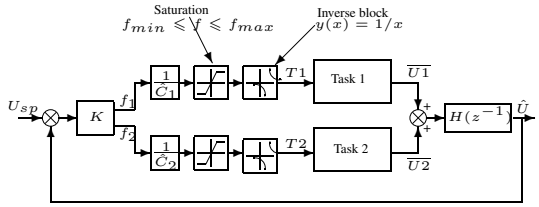

FIG. **11. Ordonnancement régulé de 2 tâches**

- L'utilisation d'un modèle linéaire pour les tâches et l'activité CPU permet d'utiliser la théorie de la commande de systèmes linéaires pour synthétiser le régulateur d'ordonnancement. Le choix d'une méthode de synthèse H_∞ permet de privilégier l'aspect robustesse ([34], [43]).
  Le contrôleur est synthétisé via des gabarits fréquentiels W_i (analogue à celui de la figure 15) permettant de spécifier simultanément une exigence de performance (par exemple le temps de réponse de la boucle de commande de l'ordonnanceur) et une spécification de robustesse (la valeur de la marge de module du système).

Contrôleurs de processus Les processus commandés sont de classiques pendules inversés. Le modèle de commande est $x(k+1) = Ax(k)+Bu(k-d)$ où x est le vecteur d'état, u le vecteur de commande, A et B des matrices de dimensions appropriées et d un délai positif, inconnu mais borné. Ce délai concatène l'ensemble des délais de boucle, provenant des durées de calcul, des préemptions et des communications. Il est montré dans [35] que l'on peut calculer une commande par retour d'état $u(k) = Kx(k)$ permettant de stabiliser le système pour tout délai d tel que $0 \leqslant d \leqslant \bar{d}$. Le problème de modélisation du retard se simplifie donc en estimation d'une valeur maximum. Par contre la performance du système, par exemple en terme de temps de réponse, n'a pas été spécifiée et va en pratique dépendre de la valeur réelle du délai : on a dans ce cas robustesse en stabilité mais pas en performance.

Le contrôleur est synthétisé par une méthode de résolution LMI (Linear Matrix Inequality), les contrôleurs obtenus sont des retours d'état sans mémoire (dont l'ordre est celui du système commandé), peu coûteux à exécuter. Le calcul de leurs gains est par contre très coûteux, il est donc effectué hors ligne pour la gamme prévue de périodes d'échantillonnage et les gains des contrôleurs correspondants sont tabulés en mémoire. On a représenté figure 12 les résultats de simulation comparant une commande classique par placement de pôles et la commande robuste présentée. Les contrôleurs robustes sont tabulés pour la gamme de périodes [0.02-0.5] s par pas de 0.02 s. Le retard maximum $\bar{d}$ admissible par le contrôleur robuste est de 0.05 s, la com-

mande par placement de pôles est synthétisée en négligeant le retard. Le contrôleur H_∞ d'ordonnancement est spécifié pour un temps de réponse de 4 sec. et une marge de module de 0,5 (valeur classique en commande robuste), sa période est fixée à 2 s.

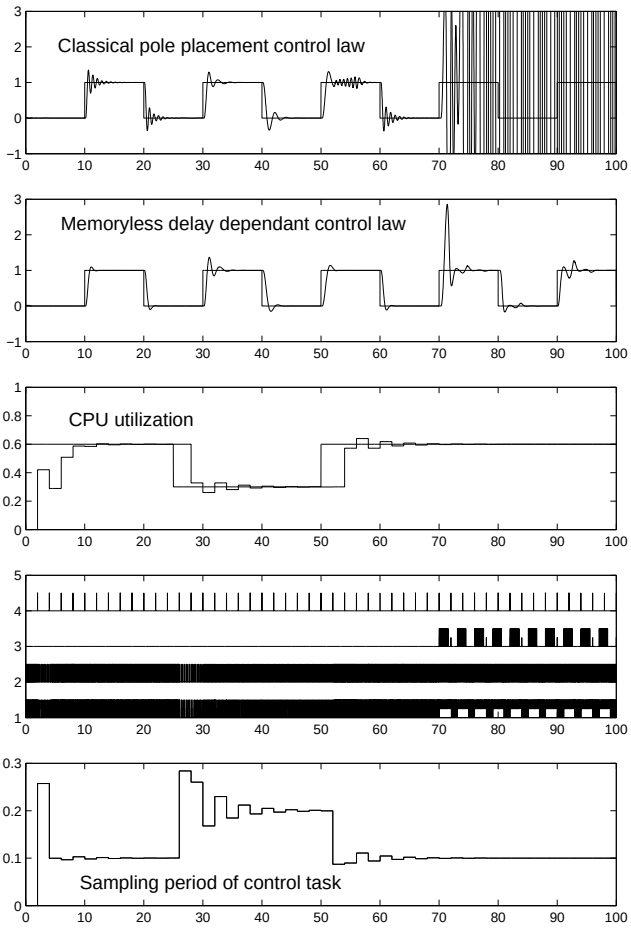

FIG. **12. Feedback scheduling + commande robuste**

En l'absence de retard les deux commandes sont stables. On constate que la charge CPU effectivement utilisée par les contrôleurs suivent la consigne de charge avec le temps de réponse spécifié.

Au temps t=70 s on admet dans le système une tâche perturbatrice prioritaire non mesurée, introduisant par préemption des contrôleurs un retard non compensé. On constate que la commande par placement de pôles est déstabilisée, contrairement à la commande robuste qui subit cependant une dégradation de performance.

On a dans cet exemple utilisé conjointement des résultats de commande robuste aux retards, permettant de contrer l'effet de délais non modélisés induits par l'implémentation, et d'ordonnancement régulé permettant d'adapter aux ressources de calcul disponibles les paramètres d'exécution de lois de commande.

6.4 RST et performances variables

Un défaut de l'exemple précédent est de ne pas permettre de spécifier explicitement les performances des processus contrôlés.

On sait que les performances d'un système de commande varient avec les paramètres d'exécution (périodes, latences et gigue), et que d'une façon générale ces performances sont dégradées quand les valeurs de ces paramètres augmentent. On peut donc supposer que lorsque les ressources d'exécution sont limitées il serait utile de revoir à la baisse les performances désirées pour les rendre compatibles avec un ordonnancement réalisable et conserver la faisabilité de l'exécution en temps-réel. La figure 6.4 compare en simulation le comportement d'un contrôleur sur la gamme de fréquences [2-50] ms : une spécification de performance (temps de réponse) constante aboutit (en haut) à une déstabilisation du système alors qu'une dégradation volontaire de la performance spécifiée, cohérente avec la période réalisable, maintient la stabilité du système.

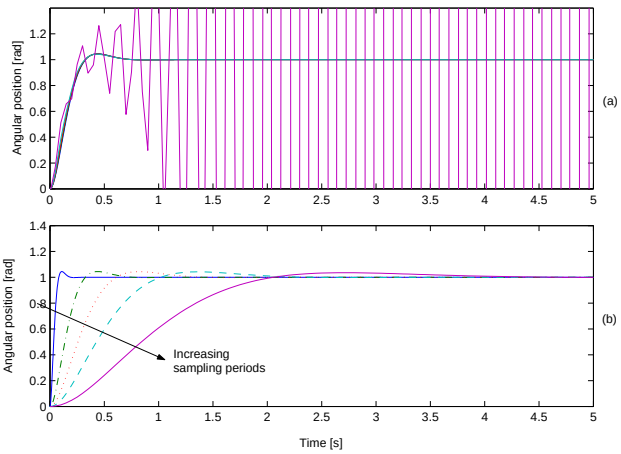

FIG. **13. Performance fixe vs. variable**

Dans [30] on utilise pour cela un régulateur de type RST (une structure générale de commande sous forme polynomiale de système mono-entrée/mono-sortie figure 14). Contrairement au cas classique où le processus est échantillonné à cadence fixe, on va ici considérer que l'on échantillonne avec une fréquence variable . La fonction de transfert en z du régulateur va être explicitement paramétrée par la période d'horloge h. Le modèle échantillonné paramétré par h est de la forme :

$$G(z,h) = \frac{B(z,h)}{A(z,h)} = \frac{b_i(h)z^i + ... + b_0(h)}{z^j + a_{j-1}(h)z^{j-1} + ... + a_0(h)}$$

Pour des systèmes mécaniques tels que les pendules utilisés dans l'exemple proposé l'ordre du système est 2. Les gains du contrôleur sont calculés de façon à ce que le système bouclé se comporte comme un système idéal de même ordre spécifié par :

$$G_{cl}(z,h) = \frac{B(z,h)\,T(z,h)}{A(z,h)\,R(z,h) + B(z,h)\,S(z,h)}$$

La résolution du problème est effectuée sous forme d'équations Diophantine ([4]), résolues de manière analytique en fonction du paramètre h. Pour conserver la stabilité cette paramétrisation implique une variation lente du paramètre h (ce qui est vrai dans ce cas où h varie au plus vite à la fréquence du régulateur d'ordonnancement).

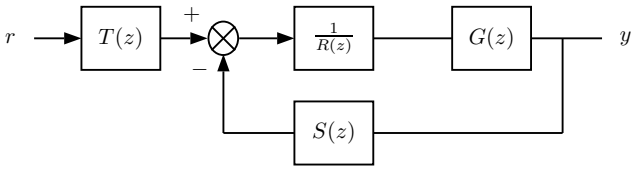

FIG. **14. Régulateur RST**

Spécification de performances Suivant une heuristique classique on choisit pour le contrôleur une période d'échantillonnage h telle que $\omega_{cl}\ h \approx 0.2 \ldots 0.6$, où ω_{cl} est la pulsation la plus rapide présente dans le système bouclé. On introduit ainsi une dépendance entre la performance de commande via ω_{cl} et la période du contrôleur que l'on trouve finalement être de la forme :

$$\begin{aligned} R(z,h) &= (z-1)\,(r_1(h)\,z + r_0(h)) \\ S(z,h) &= s_2(h)\,z^2 + s_1(h)\,z + s_0(h) \\ T(z,h) &= t_2(h)\,z^2 + t_1(h)\,z + t_0(h) \end{aligned}$$

où chaque paramètre est une fraction rationnelle de degré au plus 4, donc encore suffisamment simple pour être calculé en temps-réel.

Régulateur d'ordonnancement Comme dans l'exemple précédent, l'ordonnanceur est régulé par une commande H_∞, utilisant deux gabarits de spécification (figure 15)

- $W_e(s) = \frac{s/M_s+\omega_b}{s+\omega_s\epsilon}$ spécifie le temps de réponse du régulateur d'ordonnancement ;
- W_x spécifie l'allocation du CPU entre les tâches $\frac{U_2}{U_1} \approx \alpha$ modélisant ainsi l'importance relative des contrôleurs.

Les exemples de simulation utilisant cette méthode ([30]) montrent que l'adaptation de la spécification de performance du processus commandé aux variations de la période d'échantillonnage (elle même consécutive à une diminution de la puissance de calcul disponible) permet de gérer une dégradation progressive et contrôlée du système bouclé.

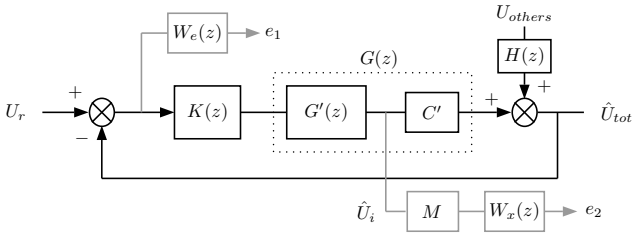

FIG. **15. Régulateur d'ordonnancement H_∞**

6.5 Mise en oeuvre : commande dynamique de robots

Les exemples précédents ont été validés uniquement en simulation, moyennant un certain nombre d'hypothèses, en particulier on suppose connaître un modèle des durées de tâches, par exemple de type constante + bruit de variance connue. On suppose également que le support d'exécution est instrumenté de façon à rendre réalisable la régulation d'ordonnancement.

Nous allons dans ce dernier paragraphe d'évaluer la faisabilité de l'ordonnancement régulé en utilisant des O.S. existants au travers d'un exemple.

Il s'agit d'implémenter la commande en position d'un bras robot (décrite en détail dans [41]).

Le modèle dynamique du bras est

$$\Gamma = M(q)\ddot{q} + Gra(q) + C(q,\dot{q})$$

où Γ est le vecteur des couples de commande, M la matrice d'inertie du bras, Gra le vecteur des forces de gravité et C le vecteur des forces de Coriolis et centrifuges. q et $\dot{q}$ sont les vecteurs des positions et vitesses articulaires du bras, de dimension 7 dans ce cas précis.

La commande non-linéaire (dite Computed Torque Controller) suivante compense par calcul explicite de modèles les variations de la matrice d'inertie ainsi que les forces parasites :

$$\Gamma = \hat{Gra}(q) + \hat{C}(q,\dot{q}) + \hat{M}[K_p(q_d - q) + K_d(\dot{q}_d - \dot{q})]$$

Elle est réalisée sous forme d'un système multi-tâches, avec Γ comme tâche de stabilisation et $\hat{M}$, $\hat{Gra}$ et $\hat{C}$ pour le calcul de termes correctifs (compensation de la dynamique non-linéaire du bras), auxquelles s'ajoute une tâche de génération de trajectoire (figure 17). Toutes ces tâches coopèrent pour la réalisation de la poursuite de trajectoire par le robot tout en étant en concurrence pour l'utilisation du calculateur.

La théorie de la commande n'est pas capable de donner une valeur optimale (si elle existe), ni même raisonnablement bonne, des périodes des tâches minimisant par exemple un critère de performance de suivi de trajectoire.

Notons que, vu le caractère non-linéaire du modèle, le poids respectif de ces périodes devrait varier suivant l'état instantané (position et vitesse) instantané du système commandé. Le but d'un ordonnanceur régulé est dans ce cas d'adapter les périodes des tâches de compensation de façon à obtenir un suivi de trajectoire raisonnablement bon, et d'éviter les overruns malgré une connaissance très approchée des fonctions de sensibilité performance/période et des durées d'exécution des tâches de calcul.

Performance et périodes On utilise comme critère de performance : $J = \int_0^{t_{trajectory}} \sum_{i=1}^{i=7} (q_i - qd_i)^2 dt$, l'intégrale du carré de l'erreur de poursuite sur une trajectoire particulière. Les fonctions de coût sont évaluées en faisant varier séparément la fréquence d'exécution de chacune des 3 tâches de compensation, les 2 autres étant pendant ce temps exécutées à 1KHz. La tâche de stabilisation P.D. (critique pour la stabilité du système) est elle toujours exécutée à une fréquence fixe de 1KHz (figure 16). Notons que, en raison de la nature non-linéaire du système commandé, ces fonctions de coût varient suivant les trajectoires réalisées en terme de positions mais aussi en vitesse de parcours. Pour la trajectoire particulière considérée dans cette expérience on a établi que $J(Iner) \approx J(Coriolis) \approx \frac{1}{2} J(Gra)$.

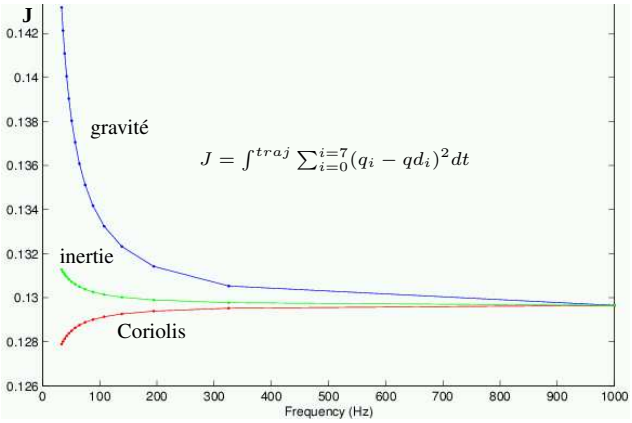

FIG. **16. Fonctions de coût J/fréquence**

Régulateur d'ordonnancement Comme pour l'exemple précédent on utilise une structure de régulateur H_∞ dans lequel les choix de l'utilisateur sont traduits au travers de deux gabarits :

- la fonction de transfert sur le signal d'écart $W_e(s) = \frac{s/M_s + \omega_b}{s + \omega_s \epsilon}$ permet de spécifier la dynamique désirée sur la poursuite de la consigne de charge CPU ;
- les gabarits W_x et M sur la variable ce commande spécifient des poids tels que

$$U_{gravity} = U_{Coriolis} + U_{inertia}$$

, i.e. la fraction de puissance de calcul allouée à chacune des tâches compensatrices est fonction de son importance relative, évaluée à partir des fonctions de coût établies au paragraphe précédent.

Le contrôleur est de la forme $Y_k = CX_k + Du_k$ où u_k est le signal d'erreur (référence - estimation de charge), Y_k (de dimension 3) est le vecteur des fréquences des tâches compensatrices Cor, Gra et $Iner$ et X est le vecteur d'état de l'ordonnanceur (de dimension 4) calculé par $X_{k+1} = AX_k + Bu_k$. A, B, C et D sont des matrices et vecteurs de paramètres constants calculés hors-ligne grâce à la toolbox H_∞ de Matlab.

Le contrôleur résultant est donc de complexité faible. Il est de plus exécuté à une fréquence plus lente que toutes les tâches de commande du robot et sa surcharge induite est donc faible.

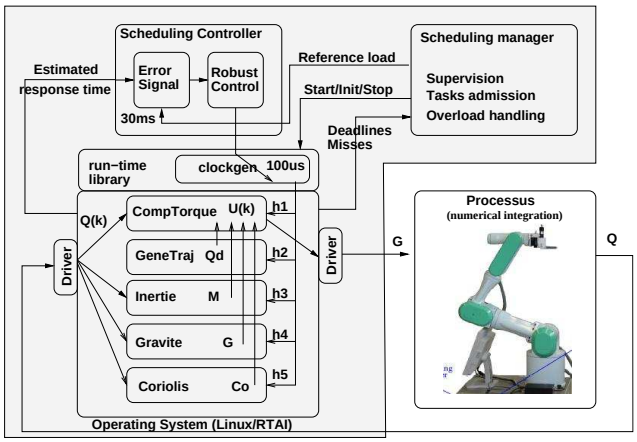

FIG. **17. Architecture de commande de robot**

Faisabilité et architecture Les contrôleurs sont implémentés sur l'architecture multi-tâches décrite par la figure 17. L'ordre décroissant des priorités fixes est déterminé par l'urgence relative des tâches est donné dans la table suivante :

$ClockGen$ (générateur d'horloge 2kHz)
$SchedulingManager$ (event driven)
$SchedulingController$ (période fixe 30 msec)
Γ (période fixe 1msec)
$GeneTraj$ (période fixe 5 msec)
$\hat{Gra} \succ \hat{Cor} \succ \hat{Iner}$ (périodes variables 1-30 msec)

Les priorités ont été affectées en tenant compte de l'urgence relative des tâches, ainsi le contrôleur ayant la période la plus grande est le régulateur d'ordonnancement, mais il contrôle le fonctionnement des contrôleurs de processus, on lui a donc affecté une priorité plus importante qu'à ceux ci. On attribue à la tâche de génération d'horloges la priorité

maximum (dans l'espace utilisateur) de façon à maximiser la régularité des horloges.

L'estimation de la durée d'estimation des modules de calcul est plus ou moins facile et précise suivant l'instrumentation dont est doté l'OS. Par exemple, la mesure est directe avec RTAI (il suffit de lire le champ exectime du descripteur de tâches mis à jour par l'ordonnanceur). Avec Linux et sa librairie de threads Posix (NPTL), il est facile d'estimer le temps de réponse d'un thread mais sans pouvoir y discerner les durées éventuelles de préemption. La norme Posix prévoit une horloge optionnelle CLOCK_THREAD_CPUTIME_ID dont l'utilisation doit permettre une mesure précise et portable de durée d'activité des threads d'une application.

Comparaisons TrueTime et Linux/RTAI Les figures 18 et 19 donnent des résultats de simulation TrueTime (à gauche) et expérimentaux sur un Pentium II 400 MHz (à droite). Le système utilisé est RTAI, permettant d'obtenir une mesure précise des durées d'exécution. À noter que, par mesure de prudence et de disponibilité, le robot (et le robot seul) est simulé : dans ce cas le driver associé à la tâche de commande appelle un intégrateur numérique et un modèle de simulation très complet du robot. Ce driver particulier a donc une durée d'exécution grande et variable, et induit une charge de calcul supplémentaire importante sur le CPU qui doit être dimensionné en conséquence.

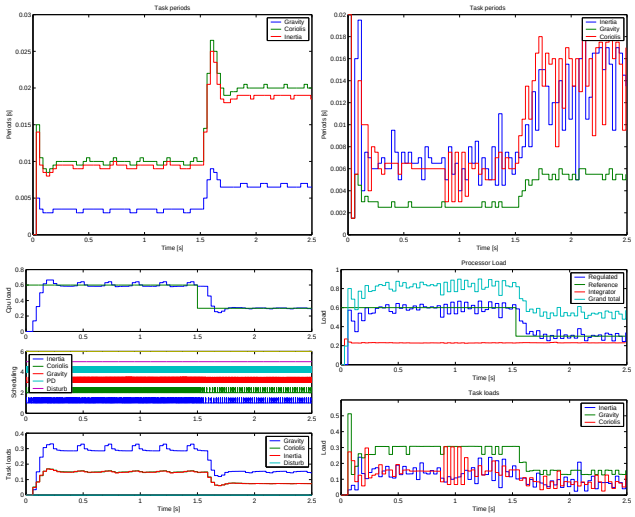

FIG. **18. Périodes et charge CPU**

Les réponses du régulateur d'ordonnancement (périodes et charges mesurées figure 18) diffèrent essentiellement par le niveau de bruit : ce bruit est dû à des variations de durées d'exécution des tâches de calcul (bien que les algorithmes soient tous à nombre d'opérations constants) et à une gigue incompressible des latences d'interruptions (chipset et contrôleur d'ITs...). On peut voir sur la figure 19 que ce bruit n'apparaît pas sur les sorties du processus (positions et vitesses articulaires), celui ci se comportant en filtre passe bas (mais le modèle de simulation ne comporte pas de modes élastiques rapides pouvant être excités par des hautes fréquences).

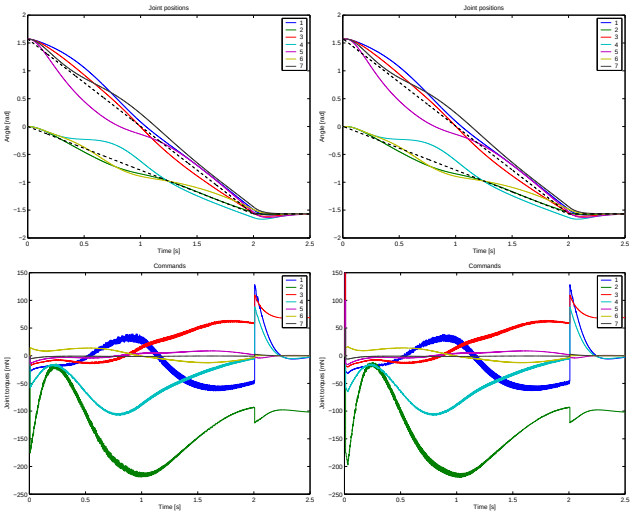

FIG. **19. Position des axes et commandes**

7 Conclusion

Les systèmes de contrôle/commande associent étroitement algorithmes de commande en boucle fermée, logiciel temps-réel embarqué et processus physiques commandés. La conception et l'implémentation de systèmes efficaces et fiables nécessite de prendre en compte des contraintes multiples et parfois contradictoires. La conception conjointe commande/ordonnancement tente de concilier dès le départ de la conception du système les problèmes de performance de commande liés aux contraintes d'implémentation. Si quelques résultats positifs ont pu être obtenus, de nombreux problèmes restent ouverts, en particulier :

- Les relations entre performance de commande et paramètres d'ordonnancement restent mal connus ; si les progrès récents en commande de systèmes à retards donnent maintenant des résultats utilisables en commande robuste, la connaissance du comportement de commandes face à des pertes occasionnelles de données reste très insuffisante pour pouvoir en tirer une formulation en terme de qualité de service et élaborer des commandes robustes et des stratégies de recouvrement d'erreurs adaptées.
- Un système de contrôle/commande distribué peut être adapté vis à vis des perturbations et incertitudes à plusieurs niveaux, pour lesquels des algorithmes efficaces restent à développer : commande spécifique-

ment robuste aux incertitudes temporelles au niveau des contrôleurs de processus (e.g. robustesse aux retards), adaptation dynamiques des paramètres d'ordonnancement au niveau intermédiaire (e.g. feedback scheduling, (m,k)firm policy) et stratégies globales de sûreté de fonctionnement et de gestion de qualité de service au niveau application. Ces différentes stratégies sont l'objet d'un compromis entre diverses contraintes et doivent être développées de façon cohérente.

– Ces stratégies adaptatives agissent en fonction d'observations faites sur l'ensemble du système (processus, calculateurs et réseau) et doivent aussi pouvoir agir sur celui ci. La plate-forme d'exécution, i.e. l'OS temps-réel et le réseau, doit être instrumentés de façon à fournir des mesures et/ou des estimations fiables et précises de l'état du système. Les mesures brutes (e.g. durée d'exécution d'un segment de code) doivent être rendues disponibles au niveau du noyau. Idéalement les mesures plus complexes (e.g. durée d'exécution d'une tâche sur une fenêtre temporelle donnée) et des actions génériques (e.g. modifier la période d'une tâche) devraient être des fonctionnalités portable fournies par une couche intergicielle (ou une API Posix ?).

Références

[1] T. Abdelzaher, E. Atkins, and K. Shin. Qos negotiation in real-time systems and its application to automated flight control. In *IEEE Real-Time Technology and Applications Symposium*, Montreal, june 1997.

[2] T. F. Abdelzaher and C. Lu. Modeling and performance control of internet servers. In *39th IEEE Conference on Decision and Control*, Sydney, Australia, december 2000.

[3] K. Arzen, B. Bernhardsson, J. Eker, A. Cervin, P. Persson, K. Nilsson, and L. Sha. Integrated control and scheduling. Technical Report ISRN LUFTD2/TFRT–7686–SE, Dpt. of Automatic Control, Lund Inst. of Technology, august 1999.

[4] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 1997.

[5] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *IEEE Trans. on Computers*, 50(4) :308–321, 2001.

[6] G. Buttazzo and L. Abeni. Adaptive rate control through elastic scheduling. In *39th Conference on Decision and Control*, Sydney, Australia, 2000.

[7] A. Cervin. Towards the integration of control and real-time scheduling design. Licentiate thesis tfrt-3226, Department of Automatic Control, Lund University, may 2000.

[8] A. Cervin. *Integrated Control and Real-Time Scheduling*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, Apr. 2003.

[9] A. Cervin and J. Eker. The Control Server : A computational model for real-time control tasks. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 113–120, Porto, Portugal, July 2003.

[10] A. Cervin, J. Eker, B. Bernhardsson, and K. Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1–2) :25–53, July 2002.

[11] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. Årzén. How does control timing affect performance ? *IEEE Control Systems Magazine*, 23(3) :16–30, June 2003.

[12] A. Cervin, B. Lincoln, J. Eker, K.-E. Årzén, and G. Buttazzo. The jitter margin and its application in the design of real-time control systems. In *10th Int. Conf. on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, Göteboeg, Sweden, august 2004.

[13] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. *Ordonnancement temps réel*. HERMES Science Publications, Paris, 2000.

[14] P. de Larminat. La commande robuste : un tour d'horizon. *RAIRO APII*, 25(3) :p 276–296, 1991.

[15] P. de Larminat. *Automatique : Commande des systèmes linéaires*. Hermes Science Publications, 1996, 2ème ed.

[16] J. Delacroix. Stabilité et régisseur d'ordonnancement en temps réel. *Technique et Science Informatiques*, 13(2) :pp223–250, 1994.

[17] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Mimo control of an apache web server : Modeling and controller design. In *American Control Conference*, Anchorage, may 2002.

[18] J. Eker and A. Cervin. A matlab toolbox for real-time and control systems co-design. In *6th International Conference on Real-Time Computing Systems and Applications*, Hong-Kong, december 1999.

[19] J. Eker, P. Hagander, and K.-E. Årzén. A feedback scheduler for real-time controller tasks. *Control Engineering Practice*, 8(12) :pp 1369–1378, 2000.

[20] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[21] D. Henriksson, A. Cervin, J. Åkesson, and K. Årzén. On dynamic realtime scheduling of model predictive controllers. In *41st IEEE Conf. on Decision and Control*, Las Vegas, 2002.

[22] A. Jaritz and M. Spong. An experimental comparison of robust control algorithms on a direct drive manipulator. *IEEE Trans. on Control Systems Technology*, 4(6), November 1996.

[23] C. Kaiser. Système d'acquisition et d'analyse en temps-réel des signaux d'un laminoir rhénalu à neuf-brisach. In *École d'été ETR'99 Applications, Réseaux et Systèmes*, ENSMA, Poitiers, Septembre 1999.

[24] Y. Lee and W. Kwon. Delay-dependent robust stabilization of uncertain discrete discrete-time state-delayed systems. In *FAC 15th World Congress*, Barcelona, Spain, 2002.

[25] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1) :46–61, 1973.

[26] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *IEEE Real-Time Technology and Applications Symposium*, Taipei,Taiwan, june 2001.

[27] C. Lu, J. Stankovic, T. Abdelzaher, G. Tao, S. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *Real-Time Systems Symposium*, december 2000.

[28] C. Lu, J. Stankovic, S. Son, and G. Tao. Feedback control real-time scheduling : Framework, modeling and algorithms. *Real-Time Systems Journal, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23(1/2) :85–126, July 2002.

[29] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Systems Journal*, 23(1-2), 2002.

[30] D. Robert, O. Sename, and D. Simon. Sampling period dependent rst controller used in control/scheduling co-d esign. In *16th IFAC 2005 World Conference*, Prague, july 2005.

[31] M. Ryu, S. Hong, and M. Saksena. Streamlining real-time controller design - from perforamance specifications to end-to-end timing constraints. In *IEEE Real-Time Technology and Applications Symposium*, Montreal, june 1997.

[32] M. Saksena, A. Ptak, P. Freedman, and P. Rodziewicz. Schedulability analysis for automated implementations of real-time object-oriented models. In *IEEE Real-Time Systems Symposium*, Madrid, december 1998.

[33] M. Sanfridson. Problem formulations for qos management in automatic control. Technical Report TRITA-MMK 2000 :3, ISSN 1400-1179, ISRN KTH/MMK–00/3–SE, KTH, Stockholm, 2000.

[34] G. Scorletti and V. Fromion. Introduction à la commande multivariable des systèmes : méthodes de synthèse fréquentielle H_∞. `www.greyc.ismra.fr/LAP/Gerard_S/ENSI_comrob.html`, 2004.

[35] O. Sename, D. Simon, and D. Robert. Feedback scheduling for real-time control of systems with communication delays. In *IEEE Conference on Emerging Technologies and Factory Automation*, Lisbon, Portugal, Sept. 2003.

[36] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *17th IEEE Real-Time Systems Symposium*, Washington, december 1996.

[37] D. Seto, J. P. Lehoczky, and L. Sha. Task period selection and schedulability in real-time systems. In *IEEE Real-Time Systems Symposium RTSS '98*, Washington DC, 1998.

[38] L. Sha, T. Abdelzaher, K.-E. årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory : A historical perspective. *Real Time Systems*, 28(2) :101–156, 2004.

[39] D. Simon and F. Benattar. Design of real-time periodic control systems through synchronisation and fixed priorities. *Int. Journal of Systems Science*, 36(2) :57–76, 2005.

[40] D. Simon, E. Castillo, and P. Freedman. Design and analysis of synchronization for real-time closed-loop control in robotics. *IEEE Transactions on Control Systems Technology*, 6(4) :pp 445–461, july 1998.

[41] D. Simon, D. Robert, and O. Sename. Robust control/scheduling co-design : application to robot control. In *RTAS'05 IEEE Real-Time and Embedded Technology and Applications Symposium*, San Francisco, march 2005.

[42] S. Skiena. *The Algorithm Design Manual*. Telos/Springer-Verlag, New York, 1997.

[43] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control : analysis and design*. John Wiley and Sons, 1996.

[44] M. Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real Time Systems*, 14(3) :219–250, 1998.