L'ordonnancement, la clé d'une gestion efficace des ressources

Nicolas Navet – Jean-Pierre Thomesse
Equipe TRIO – laboratoire LORIA à Nancy

Ordonnancer c'est définir un ordre sur l'exécution des différentes activités d'un système. Lorsqu'il s'agit d'un système informatique ces différentes activités se décomposent en l'exécution de tâches sur un ou plusieurs processeurs et éventuellement en la transmission de messages sur un réseau de communication. Tout le problème consiste à trouver le bon ordonnancement vis-à-vis des contraintes pesant sur le système, contraintes qui sont issues de la dynamique du processus physique contrôlé. Généralement il s'agira de respecter des contraintes de date au plus tard sur les fins d'exécution (ou contraintes d'échéances) mais il existe bien d'autres contraintes comme celle de date au plus tôt ou de simultanéité sur l'occurrence d'événements comme le déclenchement d'actions. Dans le contexte de l'ordonnancement, un système est dit *faisable* si toutes les invocations successives des activités qui le composent (les différentes instances) sont assurées de respecter leurs contraintes d'échéances.

1] Le paradoxe du lièvre et de la tortue (cf. [5])

Pour illustrer le fait que des capacités de traitement importantes (vitesse des processeurs, débit des réseaux) ne nous permettent pas de faire l'économie d'une réflexion sur les choix en matière d'ordonnancement, G. Le Lann eut l'idée d'utiliser le paradoxe du Lièvre et de la Tortue dans une version informatique (cf. référence [5]). Le processeur Lièvre est 10 fois plus rapide que le processeur Tortue, deux tâches A et B ont respectivement une durée d'exécution de 200 et 10 unités de temps (à vitesse 1) et une échéance fixée à t+220 (tâche A) et t+15 (tâche B) où t est la date de mise à disposition de la tâche. L'ordonnancement sur le processeur Tortue est effectuée selon la politique « plus petite échéance d'abord » (EDF - *Earliest Deadline First*) alors que le processeur Lièvre est ordonnancé sous « premier arrivé – premier servi » (*First-In - First-out*). La figure ci-dessous représente le déroulement de l'exécution sur les deux processeurs avec la tâche A qui est mise à disposition à l'instant 0 et la tâche B à l'instant 2.

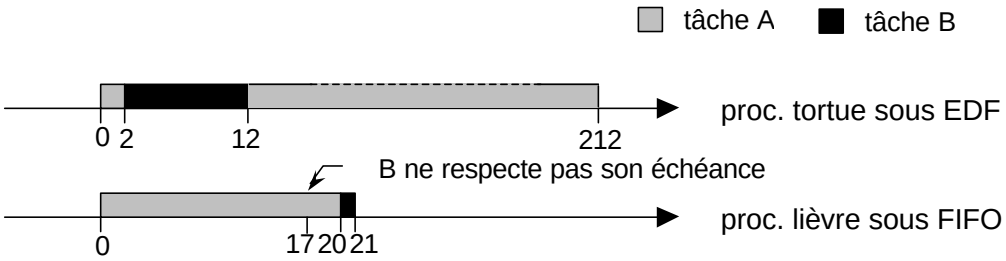

Figure 1 : Ordonnancement sur les processeurs « tortue » et « lièvre » (tiré de [5]).

Les échéances sont respectées sur le processeur Tortue et non sur le processeur Lièvre, la politique EDF est ici mieux adaptée à notre objectif de respect des échéances que FIFO (dans ce contexte, EDF a été prouvée optimale vis-à-vis des échéances, cf. [1]). D'une façon générale, des choix judicieux en matière d'ordonnancement permettront de dimensionner les ressources matérielles au plus juste tout en garantissant que le système aura les performances requises.

2] Principales approches de l'ordonnancement

La nécessité d'ordonnancer apparaît au niveau d'un processeur dès que plusieurs tâches peuvent être actives simultanément mais elle est également bien présente dans les protocoles de communication en particulier au niveau MAC (*Médium Access Control*) pour transmettre des informations sur un médium partagé. On retrouve également l'ordonnancement dans d'autres

couches comme la couche réseau et la couche transport notamment pour la gestion de trafics différentiés. Malgré quelques spécificités, comme l'impossibilité d'interrompre la transmission d'une trame sans perte d'information, les principales approches de l'ordonnancement sont identiques pour les tâches comme pour les messages et lorsque notre propos sera valable dans les deux contextes, nous parlerons d'activités s'exécutant sur une ressource.

Ordonnancement statique et périodique

On considère ici des activités strictement périodiques. L'ordonnancement de ces activités est entièrement pré-calculé hors-ligne et est stocké dans une table qui sera « déroulée » pendant l'exécution de l'application. Pratiquement, une telle table contient des couples date-activité qui indiquent la date à laquelle la ressource sera allouée à l'activité identifiée. Lorsque les activités sont périodiques, l'ordonnancement est calculé pour un PPCM des périodes puis il sera « rejoué » indéfiniment.
Un avantage certain de cette technique est que le respect des échéances du système sera garantie par construction. Quand on parle de garantie, c'est toujours sous réserve de certaines hypothèses de bon fonctionnement, en particulier de respect des durées d'exécution prévues. Un autre point positif de cette stratégie est que l'algorithme de construction de la table peut intégrer des critères autres que la faisabilité comme par exemple des contraintes de précédence entre tâche, des contraintes liées au partage de ressources, la volonté de minimiser la variabilité sur les dates de fin d'exécution d'instances successives d'une même tâche etc …
Cette technique possède néanmoins de sérieux désavantages : elle ne permet pas la construction incrémentale d'un système (l'ajout d'une activité périodique nécessite généralement de reconstruire totalement la table) et elle n'est pas adaptée pour accepter dynamiquement de la charge.
L'utilisation de l'ordonnancement statique et cyclique est répandue à la fois pour l'ordonnancement processeur (généralement dans des systèmes embarqués de taille restreinte) mais aussi dans des réseaux de communication tels que FIP (*Factory Instrumentation Protocol*), TTP/C (*Time Triggered Protocol*) et PROFIBUS (pour la gestion des stations esclaves).

Ordonnancement à priorités fixes

Nous nous situons maintenant dans le cadre de l'ordonnancement en-ligne, c'est à dire que les décisions sont prises pendant l'exécution de l'application selon un algorithme qui définit la politique d'ordonnancement. On parle d'ordonnancement à priorités fixes lorsque les différentes instances successives d'une même activité possèdent toutes la même priorité. Si la ressource est toujours allouée à l'instance qui possède la priorité la plus forte alors la politique d'ordonnancement résultante est connue dans sa version préemptive (qui permet qu'une tâche soit interrompue pendant son exécution) sous le nom de FPP (*Fixed-Priority Preemptive*).
La politique FPP possède des points forts incontestables qui en font très vraisemblablement la politique la plus couramment mise en oeuvre pour l'ordonnancement de tâches dans les applications temps réel. Cette politique a fait l'objet de très nombreuses études depuis l'article fondateur de Liu and Layland [2]. Des résultats nombreux et puissants existent tant au niveau de la faisabilité du système (temps de réponse, test de faisabilité basé sur la charge - cf. section 2) qu'au niveau du choix optimal des priorités des activités. D'autre part, dans le cadre d'un ordonnancement préemptif, si une tâche ne respecte pas les hypothèses faites lors de la conception du système (temps d'exécution trop long, fréquence d'activation trop élevée), seules des tâches de priorités inférieures seront affectées. Enfin, cette politique est implantée dans la quasi-totalité des OS temps réel et notamment ceux se conformant au standard Posix1003.1b dans lequel elle est définie sous le nom de *sched_fifo*. Dans le contexte de l'ordonnancement de messages, cette politique est sous-jacente à tous les MAC de type bus à priorités (CAN,VAN, J1850 etc..) dont l'utilisation est très répandue à l'heure actuelle.

Ordonnancement à priorités dynamiques

Lorsque les priorités des instances successives d'une même activité peuvent être différentes, on parle de priorités dynamiques (du point de vue de la tâche). Le représentant le plus important de cette classe de politiques est EDF dont l'essence est d'exécuter les instances dans l'ordre de leur urgence où le degré d'urgence est mesuré par la proximité de leur échéance. Cela implique qu'une instance ne peut utiliser la ressource que si toutes les instances d'échéances plus petites ont terminé leur exécution ou ne sont pas encore actives.

Dans le cadre de l'ordonnancement préemptif de tâches, EDF a le très grand avantage d'être optimale vis-à-vis de la faisabilité du système dans des contextes variés (cf. [1]) , c'est à dire que tout ensemble de tâches faisable sous une autre politique que EDF sera nécessairement faisable sous EDF. Le plus gros désagrément de cette politique est par contre son comportement instable en situation de surcharge : il est alors difficile de prédire quelles instances ne respecteront pas leurs échéances et les tâches les plus critiques pour l'application peuvent être affectées. Néanmoins divers mécanismes ont été proposées pour que EDF ait un comportement plus prévisible en situation de surcharge (cf. [8]).
Un autre frein à l'utilisation de la politique EDF est le fait que la grande majorité des systèmes d'exploitation temps réel ne l'implémente pas en standard alors que FPP est quasi-systématiquement proposée.

Ordonnancement conjoint de trafic à contraintes strictes et souples

Selon les conséquences du non-respect d'une contrainte temporelle, celle-ci peut être qualifiée de contrainte stricte (*hard real-time*) ou de souple (*soft real-time*). En pratique, une application comporte souvent les deux types de contraintes et l'objectif est alors généralement de minimiser, autant que faire ce peut, le temps de réponse des activités à contraintes souples tout en garantissant le respect des contraintes strictes. De nombreuses politiques ont été proposées pour répondre à ce besoin et des solutions efficaces existent à la fois sur la base d'un ordonnanceur FPP (par exemple *Dual-Priority* – cf. [3]) ou d'un ordonnanceur EDF (par exemple *Earliest Deadline Late* – cf. [4]). Le plus souvent l'idée sous-jacente de ces politiques est de repousser «le plus tard possible » les activités à contraintes strictes (tout en respectant l'échéance) si des activités à contraintes souples sont en attente d'exécution.

Ordonnancement de bout en bout

Dans des architectures réparties, il est fréquent que des traitements impliquent plusieurs sites. Considérons par exemple une application de contrôle-commande distribuée sur deux sites A et B (cf. figure 2). Le délai entre la lecture d'un capteur sur le site A et l'application de la consigne effectuée sur un site B, que nous appellerons temps de réponse de bout en bout du système informatique, peut être soumis à une contrainte de type date au plus tard. La détermination du délai maximum admissible devra naturellement tenir compte du comportement du processus physique sous contrôle, et ici en particulier, de la dynamique du système après l'application d'une consigne.

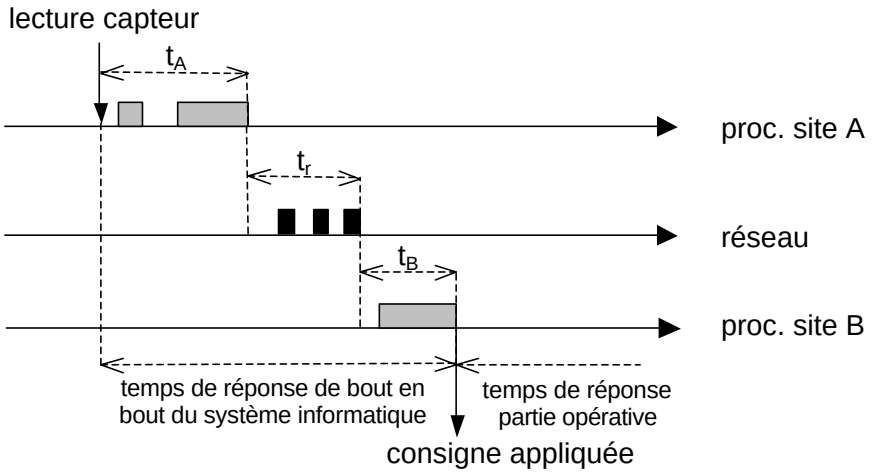

Figure 2 : Décomposition du temps de réponse de bout en bout du système informatique d'une application de contrôle-commande répartie sur deux sites.

Le temps de réponse de bout en bout du système informatique est la somme des temps de réponse sur les différentes ressources impliquées :

- sur le processeur A, il s'agit de lire la valeur en provenance du capteur, d'effectuer un éventuel traitement puis d'émettre un message à destination du site B,
- le réseau devra acheminer le message qui pourra être, dans certains cas, segmenté en plusieurs trames,
- enfin sur le site B, la consigne est calculée puis appliquée.

En plus de la puissance de traitement des ressources, les délais observés sur chacun des sites dépendront des autres activités en concurrence, des politiques d'ordonnancement et de certains choix de conception comme le fait d'invoquer les tâches sur l'occurrence d'un événement ou de façon périodique. Ces problèmes d'ordonnancement sur des ressources en série ont été bien étudiés et des analyses existent lorsque EDF ou FPP est utilisée comme politique d'ordonnancement processeur sur les différents sites (cf. [9] pour FPP et [1] pour EDF).

3] Comment vérifier le respect des contraintes ? l'analyse d'ordonnançabilité

L'analyse d'ordonnançabilité consiste à étudier la faisabilité du système. Il s'agit de vérifier que toutes les instances des activités respectent leurs contraintes sous une politique d'ordonnancement donnée avec des paramètres fixés (priorités des tâches sous FPP, durée d'un quantum de temps sous une politique de type Round-Robin ..). En pratique, une analyse d'ordonnançabilité est soit un calcul explicite de bornes sur les temps de réponse, soit un test de faisabilité parfois simplement basé sur la charge globale du système comme c'est le cas pour les politiques FPP et EDF. L'intérêt majeur de ces tests de faisabilité tient en leur complexité algorithmique restreinte: ils peuvent être linéaires en le nombre de tâches là où un calcul explicite de temps de réponse a une complexité polynomiale. Par contre ils ne sont en général que suffisants ; certains ensembles de tâches parfaitement faisables peuvent ne pas passer le test.

L'analyse d'ordonnançabilité ne résout pas à elle seule tous les problèmes de conception liés à l'ordonnancement, en particulier

- elle ne répond pas à la question cruciale du choix de la ou des politiques d'ordonnancement et de leurs paramètres. Une analyse d'ordonnançabilité ne propose pas de solutions faisables, elle se prononce simplement sur la faisabilité d'une allocation de politiques et de priorités qui lui aura été soumise,
- elle est d'une aide limitée pour l'optimisation du système à cause des hypothèses nécessairement pessimistes sur les pires temps d'exécution et puisque la seule trajectoire du système qui est étudiée est la plus pessimiste en termes de flux d'arrivée de travail.

Si par exemple le travail d'une tâche doit être rendu le plus régulièrement possible, il est souhaitable de minimiser la variabilité (la « gigue ») sur ses dates de fins d'exécution. L'allocation de priorités et de politique influera naturellement sur la qualité du système vis-à-vis de ce critère. Pour décider des bonnes politiques d'ordonnancement et de leurs paramètres, il est parfois nécessaire de faire appel à des techniques d'optimisation comme les algorithmes génétiques compte tenu de la complexité algorithmique des problèmes. Néanmoins il faudra toujours faire appel à l'analyse d'ordonnançabilité pour distinguer les configurations faisables des configurations non-faisables et utiliser la simulation ou des mesures sur plate-formes pour évaluer la qualité de chaque configuration faisable. Le lecteur pourra par exemple consulter [6] et [7].

Les analyses d'ordonnançabilité sont dans la littérature souvent qualifiées d'analyse du pire-cas (*worst-case schedulability analysis*) et il est parfois écrit qu'elles donnent des garanties absolues sur les temps de réponse. Si en effet les hypothèses faites en termes de quantité d'arrivée de travail sont pessimistes, il faut bien être conscient que ces analyses ne considèrent absolument pas la possibilité d'occurrence de divers aléas qui peuvent toujours survenir dans un système réel, comme des erreurs de transmission sur un réseau ou la défaillance d'un composant matériel. Si le système est critique, dans le sens où le non respect d'une de ses spécifications pendant son exécution peut avoir des conséquences graves, alors le concepteur devra étudier son système sous le jour de la sûreté de fonctionnement pour éventuellement y introduire des mécanismes de tolérance aux fautes (redondance matérielle, mécanismes protocolaires …). Si quelques études (cf. [10]) ont eu pour objectif d'intégrer des considérations de sûreté de fonctionnement aux analyses classiques de performances temporelles dans le pire cas, beaucoup de travail reste à fournir dans ce domaine.

Conclusion

La parabole du Lièvre et de la Tortue permet de réaliser que les problèmes d'ordonnancement ne se règlent pas uniquement en considérant la puissance des ressources d'autant plus que le dimensionnement des ressources au plus juste est généralement un impératif économique.

Nous avons fait un tour d'horizon des techniques d'ordonnancement les plus couramment utilisées en évoquant leurs avantages et inconvénients majeurs. Il est clair que bien d'autres politiques peuvent se justifier vis-à-vis de certains objectifs particulier comme par exemple maximiser le « throughput » (on pourra alors utiliser la politique *Shortest Processing Time First*) ou répartir équitablement la ressource (une politique de type Round-Robin est bien adaptée).
Dans la littérature technique, l'ordonnancement est souvent trop réduit aux analyses d'ordonnançabilité qui certes, sont des outils importants mais qui ne nous affranchissent pas de l'étude des répercussions des choix faits en matière d'ordonnancement sur le processus contrôlé. D'autre part, une réflexion sur l'ordonnancement s'insère dans une démarche de conception globale qui ne pourra faire abstraction, pour les systèmes les plus critiques, des questions de sûreté de fonctionnement.

[1] J. Stankovic, M. Spuri, K. Ramamritham et G. Buttazzo. *"Deadline scheduling for real-time systems"*, Kluwer Academic Publisher, 1998.

[2] C.L. Liu et J.W. Layland. *"Scheduling algorithms for multiprogramming in hard real-time environment"*. Journal of the ACM, 20(1):40-61, 1973.

[3] B. Gaujal, N. Navet et J. Migge. "*Dual-priority versus background scheduling: a path-wise comparison*". Acceptée pour publication dans Real-Time Systems, disponible à l'url http://www.loria.fr/~nnavet , 2001.

[4] H. Chetto et M. Chetto. *"Some results of the earliest deadline scheduling algorithm"*. IEEE Transactions on Software Engineering, 15(10):1261-1269, 1989.

[5] G. Le Lann. "*Critical issues for the development of distributed real-time computing systems*", Rapport de recherche INRIA n°1274, 1990.

[6] N. Navet et J. Migge. "*Fine tuning the scheduling of tasks through a genetic algorithm: application to Posix1003.1b compliant OS*". Acceptée pour publication dans IEE Proceedings Software, disponible à l'url http://www.loria.fr/~nnavet , 2002.

[7] M. DiNatale et J.A. Stankovic. *"Applicability of simulated annealing methods to real-time scheduling and jitter control*". In Proceedings of the 16th IEEE Real-Time Systems Symposium, 1995.

[8] J. Delacroix. *"Towards a stable Earliest Deadline Algorithm",* Real-Time Systems, 10(3), 263-291, 1996.

[9] K. Tindell, A. Burns et A.J. Wellings. "*An extendible approach for analysing fixed priority hard real-time sytems*", Real-Time Systems, 6(2), 1994.

[10] N. Navet, Y.-Q. Song et F. Simonot. *"Worst-case deadline failure probability in real-time applications distributed over CAN"*, Journal of Systems Architecture, 46(7), 2000, disponible à l'url http://www.loria.fr/~nnavet .

Glossaire:

- **Faisabilité** : capacité de toutes les instances d'un ensemble d'activités récurrentes à respecter leurs contraintes temporelles. Il s'agira en général de contraintes d'échéances. On dit d'un système qu'il est faisable ou ordonnançable.
- **Analyse d'ordonnançabilité** : calcul qui permet de se prononcer sur la faisabilité d'un ensemble d'activités. En pratique, il s'agit généralement d'identifier et d'étudier la trajectoire du système la plus pessimiste vis-à-vis des contraintes à vérifier.
- **Politique d'ordonnancement** : algorithme qui décide de l'attribution d'une ressource. Cet algorithme peut être exécuté pendant l'exécution de l'application (on parle alors d'ordonnancement en-ligne) mais l'ordonnancement peut-être précalculé hors-ligne et «déroulé» pendant l'exécution.

- **FPP (Fixed Priority Preemptive)** : politique d'ordonnancement de tâches sous laquelle les différentes instances successives d'une tâche récurrente possèdent toutes la même priorité et qui, à tout instant, attribue le processeur à la priorité la plus forte. Dans sa variante non-préemptive, cette politique est aujourd'hui largement utilisé dans des MAC comme CAN.
- **EDF (Earliest Deadline First)** : politique d'ordonnancement de tâches qui, à tout instant, alloue le processeur à l'instance de tâches dont l'échéance est la plus proche. Cette politique est la meilleure du point de vue de la faisabilité et permet le taux d'utilisation processeur le plus élevé.