Ordonnancement non-clairvoyant: petites simplifications et améliorations de l'analyse de la famille d'algorithmes LAPSβ

Julien Robert, Nicolas Schabanel

Ordonnancement non-clairvoyant: petites simplifications et améliorations de l'analyse de la famille d'algorithmes LAPS$_\beta$

Julien Robert[1] and Nicolas Schabanel[1] †

[1] *Université de Lyon, École normale supérieure de Lyon, LIP (UMR n°5668), France.*
[2] *CNRS, Université Paris Diderot, LIAFA (UMR n°7089), France.*

En 1999, Edmonds [1] introduit un modèle très général de tâches qui traversent différentes phases ayant différentes quantités de travail et capacités à être parallélisées. La force du modèle d'Edmonds est qu'il démontra que même si l'ordonnanceur ne connaît *strictement rien* des caractéristiques des tâches qu'il est en train d'ordonnancer et est seulement informé de leur arrivée à leur arrivée et de leur complétion à leur complétion, **EQUI**, qui partage de manière égale les processeurs entre les tâches actives, réussit à être compétitif avec l'ordonnancement optimal hors-line clairvoyant, pour peu qu'**EQUI** dispose d'un peu plus de deux fois plus de ressources que l'optimum. Ceci signifie que l'ordonnanceur **EQUI** supporte sans diverger toute charge inférieure à 50%. Nous [4] avons par la suite étendu l'analyse d'Edmonds au cas où les tâches sont composées d'un DAG de processus traversant des phases arbitraires et démontré que l'algorithme non-clairvoyant **EQUI**$\circ$**EQUI** supporte dans ce cas également toute charge inférieure à 50%. En 2009, Edmonds et Pruhs [2] ont proposé une nouvelle famille d'algorithmes $\mathbf{LAPS}_{\beta,\, 0<\beta\leqslant 1}$ qui partagent de manière égale les processeurs entre les tâches de la fraction β des tâches actives arrivées les plus récemment, et ont démontré que ces algorithmes sont $(1+\beta+\varepsilon)$-speed $\frac{4(1+\beta+\varepsilon)}{\beta\varepsilon}$-compétitif, c.-à-d. supportent une charge arbitrairement proche de 1 ($<1-\beta$) sans diverger. Ce résultat repose sur une réduction à des phases uniquement séquentielles ou parallèles. Nous utilisons ici des lemmes de [3, 4] qui nous permettent ici de gagner un facteur 2 sur le facteur de compétitivité et de simplifier la preuve, démontrant ainsi que $\mathbf{LAPS}_\beta$ est en fait $(1+\beta+\varepsilon)$-speed $\frac{2(1+\beta+\varepsilon)}{\beta\varepsilon}$-compétitif.

1 Introduction

Avec l'avénement des ordinateurs multi-core et l'incapacité actuelle à développer des algorithmes tirant réellement parti du parallélisme (à l'exception de certaines applications, vidéo par exemple), les systèmes d'exploitation vont très bientôt devoir ordonnancer des tâches qui tireront avantage de façon très aléatoire et imprévisibles des processeurs qu'on leur attribue, et ce de façon très inégale au fur et à mesure de leur avancement. Aussi il est très important de définir un cadre *réaliste* où l'ordonnanceur ne dispose d'aucune information précise sur les tâches, afin d'obtenir des algorithmes utilisables en pratique et dont l'efficacité est prouvée mathématiquement. Edmonds en 1999 [1] a donc proposé le problème suivant pour modéliser cette situation. Il s'agit d'ordonnancer *n tâches* $J_1,\ldots,J_n$ qui arrivent au cours du temps, sur p processeurs que l'on peut découper arbitrairement (la préemption est modélisée par l'attribution d'une proportion de processeurs à chaque tâche) de façon à minimiser le temps de service moyen de ces tâches (appelé *temps de flot*). Chaque tâche consiste en une suite de *phases* qui tirent plus ou moins bien parti du nombre de processeurs qu'on leur attribue. Précisément :

Les tâches. On considère une suite de *n tâches* $J_1,\ldots,J_n$ arrivant à des dates $r_1,\ldots,r_n$ inconnues de l'ordonnanceur qui doit les ordonnancer sur p processeurs. Chaque tâche J_i traverse un certain nombre de *phases* $J_{i,1},\ldots,J_{i,q_i}$ inconnues de l'ordonnanceur. Chaque phase J_{ij} est caractérisée par une quantité de *travail* w_{ij} à effectuer et un *profil d'accélération* $\Gamma_{ij}:\mathbb{R}_+\to\mathbb{R}_+$, tous deux inconnus de l'ordonnanceur

† Ce travail a été partiellement financé par l'ANR ALADDIN.

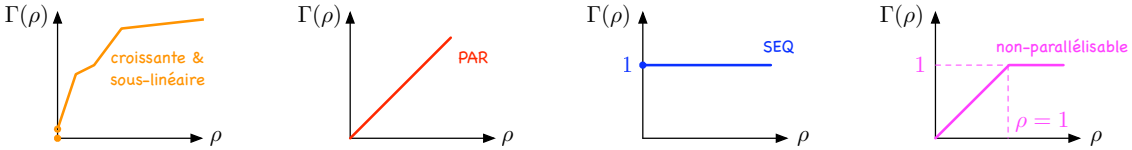

FIGURE 1: Exemples de profils d'accélération : quelconque, PAR, SEQ, et non-parallélisable.

également. Si l'ordonnanceur attribue pendant un temps dt, ρ_i processeurs ($\rho_i \in \mathbb{R}_+$) à la tâche J_i dans sa j-ème phase, le travail effectué de cette phase est : $dw_{ij} = \Gamma_{ij}(\rho_i)dt$.

On suppose de manière naturelle que chaque profil d'accélération est :

- *croissant* ($\Gamma(\rho) \leqslant \Gamma(\rho')$ si $\rho \leqslant \rho'$), c.-à-d. aucune tâche n'avance moins vite avec plus de processeurs ;
- *sous-linéaire* ($\frac{\Gamma(\rho)}{\rho} \geqslant \frac{\Gamma(\rho')}{\rho'}$ si $\rho \leqslant \rho'$), c.-à-d. aucune tâche n'utilise de manière plus efficace plus de processeurs, p. ex., en supposant qu'aucune tâche ne tire trop avantage des caches locaux.

Voici quelques exemples de profils (cf. Fig. 1). Une phase *totalement parallèle*, notée PAR, est définie par $\Gamma(\rho) = \rho$, et progresse donc de ρdt pendant dt ; une phase *séquentielle*, notée SEQ, (idle, p. ex.) est définie par $\Gamma_{ij}(\rho) = 1$ pour tout ρ (même nul), et progresse donc de dt *quelque soit* le nombre de processeurs attribués par l'ordonnanceur à la tâche, même si l'ordonnanceur ne lui en attribue aucun ; enfin $\Gamma(\rho) = \min(\rho, 1)$ définit une phase non-parallélisable qui avance proportionnellement au temps qu'on lui accorde sur un processeur, mais ne tire aucun profit de deux processeurs ou plus. Nous verrons à la proposition 1 que les phases PAR et SEQ sont les pires cas possibles pour un ordonnanceur non-clairvoyant : en effet, l'une tire parti au mieux de chaque pourcentage de processeur qu'on lui donne, alors qu'on donne des processeurs en pure perte à l'autre, tandis que l'algorithme est incapable de distinguer qui est dans quelle phase.

Ordonnancement. Un ordonnancement $\mathcal{S}_p$ sur p processeur est un n-uplet de fonctions positives constantes par morceaux $\rho_i : t \mapsto \rho_i^t$, avec $1 \leqslant i \leqslant n$, où ρ_i^t ($\in \mathbb{R}_+$) représente la quantité de processeurs allouée à la tâche J_i au temps t, telles que : pour tout t, $\sum_{i=1}^n \rho_i^t \leqslant p$. Notons c_{ij} la date de la terminaison de la j-ème phase de la tâche i. En posant $c_{i0} = r_i$, c_{ij} est défini naturellement de façon récursive par :

$$c_{ij} = \min\left\{ t' : w_{ij} = \int_{c_{i,j-1}}^{t'} dw_{ij} =_{\text{def}} \int_{c_{i,j-1}}^{t'} \Gamma_{ij}(\rho_i^t)\, dt \right\}.$$

La tâche J_i termine à la date $C_i = c_{i,q_i}$ de la complétion de sa dernière phase. La tâche J_i est *active* à partir du moment où elle apparaît ($t = r_i$) jusqu'à sa terminaison ($t = C_i$). Un ordonnancement est *valide* s'il termine bien toutes les tâches (c.-à-d., si $C_i < \infty$ pour tout i).

Fonction objectif. Nous cherchons à minimiser le temps de service moyen, ou bien de façon équivalente du point de vue de l'approximation, le temps de service total, appelé également *temps de flot* :

$$\text{Flowtime}(\mathcal{S}_p) = \sum_{i=1}^{n} (C_i - r_i) =_{\text{de manière équivalente}} \int_0^\infty (\text{nombre de tâches actives au temps } t)\, dt.$$

Non-clairvoyance, compétitivité et augmentation de ressources. Nous souhaitons un ordonnanceur réaliste, c.-à-d., *non-clairvoyant* : l'ordonnanceur découvre les tâches au moment de leur arrivée, ne connaît rien des différentes phases qu'elles traversent (rien sur leur quantité de travail, ni rien sur leur profil d'accélération) ni de leur avancement, il est seulement informé de leur terminaison au moment de leur terminaison : il sait donc *seulement* quelles tâches sont actives à chaque instant. Dans cette situation, il est évidemment difficile d'être compétitif avec l'ordonnancement hors-ligne optimal et Edmonds [1] a démontré que le temps de flot de tout algorithme non-clairvoyant (même randomisé) est nécessairement à un facteur $\Omega(n)$ du temps de flot optimal hors-ligne en présence de tâches SEQ et PAR. Aussi, Edmonds propose d'augmenter les ressources disponibles et de comparer l'ordonnancement engendré par un algorithme A sur sp processeurs à l'ordonnancement optimal sur p processeurs seulement, avec $s > 1$. On note alors $A_{sp}(J)$ l'ordonnancement engendré par A sur l'instance J avec sp processeur. On dit alors qu'un algorithme A est *s-speed c-compétitif* si pour toute instance J, $\text{Flowtime}(A_{sp}(J)) \leqslant c \cdot \text{OPT}_p(J)$, où $\text{OPT}_p(J)$ désigne le temps de flot optimal sur p processeurs. Intuitivement, A est s-speed $O(1)$-compétitif, signifie

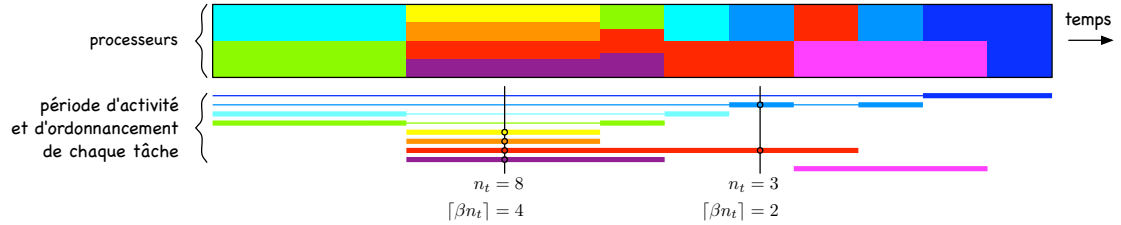

FIGURE 2: Une exécution de l'algorithme $\mathbf{LAPS}_{\beta=\frac{1}{2}}$: à chaque instant t, seules les $\lceil \frac{n_t}{2} \rceil$ tâches actives les plus récentes sont ordonnancées et se partagent équitablement les processeurs.

que A *peut tenir une charge* $1/s$, au sens que le temps de service de A ne s'éloigne pas du temps optimum "rempli à une hauteur $1/s$ seulement" quand n augmente ; ainsi tant que la charge reste inférieure à $1/s$, ses performances *ne se dégradent pas avec le temps*.

Comme l'ordonnanceur est autorisé à attribuer des fractions de processeurs, quitte à renormaliser les profils d'accélération d'un facteur $1/p$, on suppose sans perte de généralité que $p = 1$.

Les algorithmes $\mathbf{LAPS}_\beta$. Edmonds et Pruhs [2] ont démontré que les algorithmes $\mathbf{LAPS}_{\beta,0<\beta\leqslant 1}$ sont s-speed $\frac{4s}{\beta\varepsilon}$-compétitif pour tout $\varepsilon > 0$ et toute vitesse $s \geqslant 1+\beta+\varepsilon$ et donc *peuvent supporter une charge arbitrairement proche de* 1 en prenant β suffisamment petit. Notons n_t le nombre de tâches actives au temps t. L'algorithme $\mathbf{LAPS}_\beta$ procède ainsi sur s processeurs (Fig. 2) : il les partage à chaque instant de manière égale entre les $\lceil \beta n_t \rceil$ tâches actives les plus récemment arrivées, en attribuant à chacune $\frac{s}{\lceil \beta n_t \rceil}$ processeurs et rien aux $n_t - \lceil \beta n_t \rceil$ tâches plus anciennes. Remarquons que $\mathbf{EQUI} = \mathbf{LAPS}_{\beta=1}$.

2 Analyse simplifiée et améliorée de $\mathbf{LAPS}_\beta$

Nous allons démontrer ici que $\mathbf{LAPS}_{\beta,0<\beta\leqslant 1}$ est en fait s-speed $\frac{2s}{\beta\varepsilon}$-compétitif au lieu de $\frac{4s}{\beta\varepsilon}$-compétitif, pour tout $\varepsilon > 0$ et toute vitesse $s \geqslant 1+\beta+\varepsilon$. Cette amélioration réside dans une réduction plus précise aux instances SEQ-PAR.

Proposition 1 (Réduction aux instances SEQ-PAR avec retard sur le PAR)**.** *Pour toute instance $J_1, \ldots, J_n$ et tout algorithme non-clairvoyant A sur s processeurs et tout ordonnancement valide* OPT *sur* 1 *processeur, il existe une instance $J'_1, \ldots, J'_n$ de tâches consituées uniquement de phases* SEQ *ou* PAR*, telles que :*

1. *$A_s(J') = A_s(J)[J'/J]$, c.-à-d. A construit le même ordonnancement pour J' que pour J ;*
2. *$\mathrm{OPT}[J'/J]$ est un ordonnancement valide de J', ainsi $\mathrm{Flowtime}(\mathrm{OPT}[J'/J]) \leqslant \mathrm{Flowtime}(\mathrm{OPT})$;*
3. *$A_s(J')$ est toujours en retard par rapport à $\mathrm{OPT}[J'/J]$ sur tout travail parallèle dans J' ;*

où $\mathcal{S}[J'/J]$ désigne l'ordonnancement obtenu en substituant chaque tâche J_i par la tâche J'_i dans $\mathcal{S}$.

Démonstration. La preuve reprend les idées de [1] tout comme [2] mais utilise [3, 4] pour garantir le point 3. qui permettra de simplifier la preuve et d'améliorer l'analyse d'un facteur 2 par la suite.

Comme A est non-clairvoyant, on va modifier les phases que traversent chaque tâche J_i pour assurer les trois propriétés sans modifier sa période d'activité dans A. Considérons la tâche J_1 (idem pour les autres). La preuve procède en trois étapes. Premièrement, on redécoupe chaque tâche en un nombre fini de segments $K_1, \ldots, K_m$ tels que chaque segment correspond à un morceau d'une phase de J_1 qui reçoit tout du long la même quantité de processeurs dans A_s et la même quantité de processeurs dans OPT (pour cela, il suffit de noter à quel avancement de la tâche correspond chaque changement du nombre de processeurs alloués par A_s et par OPT, Fig. 3❶). Deuxièmement, pour chaque segment K_j de profil Γ, de quantité de travail w, de longueur T_A (resp. T_O) et recevant ρ_A (resp. ρ_O) processeurs dans A_s (resp. OPT), on remplace la phase par :

- une phase SEQ de longueur $w' = T_A$, si $\rho_A \geqslant \rho_O$; comme Γ est croissante, alors $w = \Gamma(\rho_A)T_A = \Gamma(\rho_O)T_O$ implique que $T_O \geqslant T_A$ et cette nouvelle phase SEQ peut donc être terminée dans l'emplacement qui était alloué à K_j dans OPT. (v. Fig. 3❷)
- une phase PAR de travail $w' = w = \rho_A T_A$ sinon ; comme Γ est sous-linéaire et $w = \Gamma(\rho_A)T_A = \Gamma(\rho_O)T_O$, $\rho_A < \rho_O$ implique que $\rho_A T_A \leqslant \Gamma(\rho_A)\frac{\rho_O}{\Gamma(\rho_O)}T_A = \rho_O T_O$, donc cette nouvelle phase PAR peut bien être terminée dans l'emplacement qui était alloué à K_j dans OPT (v. Fig. 3❷). Ainsi, on a bien le point 2.

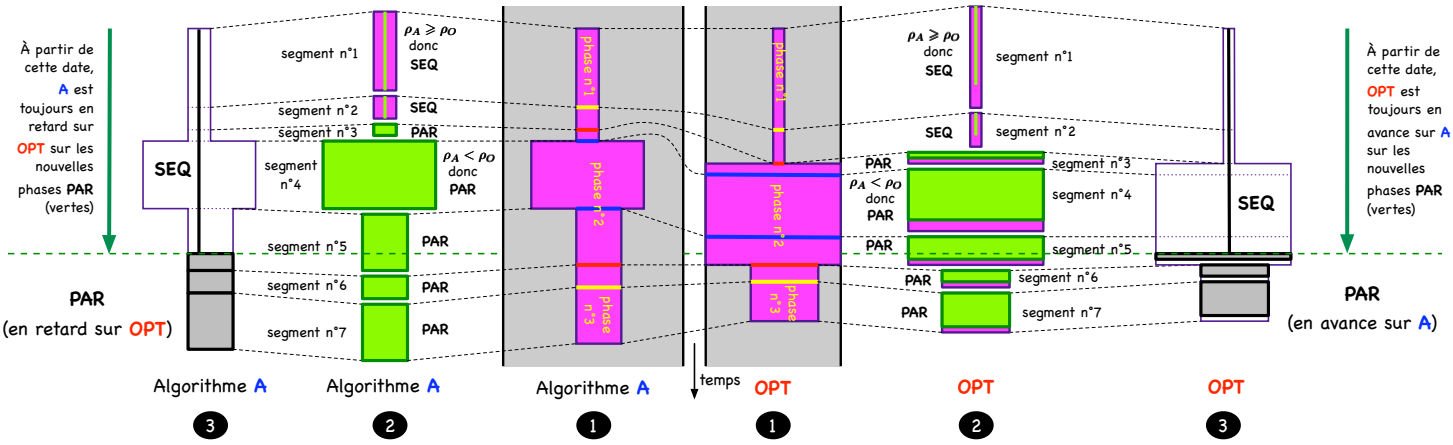

FIGURE 3: Illustration de la réduction à une instance SEQ-PAR où A exécute toujours le travail PAR en retard sur OPT.

De plus, comme A est non-clairvoyant et que les phases sont calculées pour terminer exactement au même moment dans A_s, l'ordonnancement calculé par A_s sur cette nouvelle instance est identique à celui calculé sur J, d'où le point 1. Enfin, si $A_s(J')$ exécutait quelque travail parallèle en avance sur $\mathrm{OPT}[J'/J]$, alors on recherche le dernier instant T où cela se produit, $\mathrm{OPT}[J'/J]$ et $A_s(J')$ en sont nécessairement au *même point* dans la tâche J'_1 arrivée à la même date r_1 et on remplace toutes les phases avant ce point par une unique phase SEQ de longueur $T - r_1$ (v. Fig 3❸), cela ne change rien pour A_s ni pour $\mathrm{OPT}[J'/J]$ d'où le point 3. □

Ainsi, on peut considérer que les tâches $J_1, \ldots, J_n$ sont toujours de type SEQ-PAR et que $\mathbf{LAPS}_{\beta,s}$ est toujours en retard sur les phases parallèles par rapport à OPT_1. Notons n_t le nombre de tâches actives dans $\mathbf{LAPS}_{\beta,s}$ au temps t et renumérotons de 1 à n_t ces tâches actives de la plus ancienne à la plus récente. Soit x_i^t la quantité de travail parallèle en retard de $\mathbf{LAPS}_{\beta,s}$ sur OPT_1 pour la i-ème tâche active dans cet ordre. Notons ℓ_t le nombre de tâches actives dans une phase SEQ dans $\mathbf{LAPS}_{\beta,s}$ au temps t.

Définissons le *potentiel* suivant : $\Phi(t) = \frac{2}{\varepsilon}\sum_{i=1}^{n_t} i \cdot x_i^t$ (contrairement à [2], nous n'avons pas besoin d'utiliser $\max(0, x_i^t)$ car x_i^t est toujours positif). Clairement, $\Phi(0) = \Phi(\infty) = 0$. Par construction, $\Phi(t)$ est inchangée lorsqu'une nouvelle tâche arrive, et ne peut que décroître lorsqu'une tâche se termine dans $\mathbf{LAPS}_{\beta,s}$ ou dans OPT_1. On peut maintenant conclure, en reprenant les calculs de [2] :

Lemme 2. *Si $s \geqslant 1+\beta+\varepsilon$, pour tout t où aucune tâche n'arrive ou se termine, $n_t + \frac{d\Phi(t)}{dt} \leqslant \frac{2s}{\beta\varepsilon}\ell_t$.*

Théorème 3 ([2] amélioré)**.** *$\mathbf{LAPS}_{\beta,s}$ est s-speed $\frac{2s}{\beta\varepsilon}$-compétitif pour tout $s \geqslant 1+\beta+\varepsilon$.*

Démonstration. $\mathrm{Flowtime}(\mathbf{LAPS}_{\beta,s}) = \int_0^\infty n_t dt = \int_0^\infty n_t dt + \Phi(\infty) - \Phi(0) = \int_{t:\text{pas arrivée ni fin}} (n_t + \frac{d\Phi(t)}{dt})dt + \int_{t:\text{arrivée ou fin}} \frac{d\Phi(t)}{dt}(\leqslant 0)\,dt \leqslant \frac{2s}{\beta\varepsilon}\int_0^\infty \ell_t dt = \frac{2s}{\beta\varepsilon}\sum_{i=1}^{n} \mathrm{SEQ}(J_i) \leqslant \frac{2s}{\beta\varepsilon}\mathrm{Flowtime}(\mathrm{OPT})$, où $\mathrm{SEQ}(J_i)$ désigne la somme des longueurs des phases SEQ de J_i qui est clairement une borne inférieure sur le temps de flot de J_i dans tout ordonnancement. □

Références

[1] Jeff Edmonds. Scheduling in the dark. *Manuel Blum's Special Issue of Theoretical Computer Science*, 235(1) :109–141, 2000. Version préliminaire publiée à STOC 1999.

[2] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *Proc. of ACM/SIAM Symp. On Discrete Algorithms (SODA)*, 2009. To be published. `www.cse.yorku.ca/~jeff/research/schedule/laps.pdf`.

[3] Julien Robert and Nicolas Schabanel. Non-clairvoyant batch set scheduling : Fairness is fair enough. In *Proc. of European Symp. on Algorithms (ESA)*, volume LNCS 4698, pages 742–753, Oct. 2007.

[4] Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *Proc. of ACM/SIAM Symp. On Discrete Algorithms (SODA)*, pages 491–500, 2008.