HAL
open science

Ordonnancement non-clair voyant avec dépendances : analyse de LAPS_$\beta \circ$ EQUI

Julien Robert, Nicolas Schabanel

To cite this version:

Julien Robert, Nicolas Schabanel. Ordonnancement non-clair voyant avec dépendances : analyse de LAPS_$\beta \circ$ EQUI. AlgoTel, 2009, Carry-Le-Rouet, France. hal-00383347

HAL Id: hal-00383347

https://hal.science/hal-00383347

Submitted on 12 May 2009

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.

Ordonnancement non-clairvoyant avec dépendances: analyse de $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$

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 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 [SODA'08] 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 $\mathbf{EQUI} \circ \mathbf{EQUI}$ supporte dans ce cas également toute charge inférieure à 50%. En 2009, Edmonds et Pruhs ont proposé une nouvelle famille d'algorithmes $\mathbf{LAPS}_\beta$ qui partagent de manière égale les processeurs entre les tâches de la proportion 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. À l'aide des mêmes outils que ceux que nous avions développés pour la preuve de la compétitivité de $\mathbf{EQUI} \circ \mathbf{EQUI}$ en présence de dépendances, nous montrons ici que l'algorithme $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$, composition naturelle de $\mathbf{LAPS}_\beta$ et d'**EQUI**, est $(1+\beta+\varepsilon)$-speed $\frac{(k+1)(1+\beta+\varepsilon)}{\beta\varepsilon}$-compétitif, où k est la taille maximale d'une anti-chaîne d'une tâche.

1 Introduction

Dans un centre de calcul, les tâches à exécuter sont constituées de plusieurs processus, dépendants les uns des autres. Ces processus doivent être ordonnancés en-ligne, c.-à-d. au fur et à mesure des requête pour l'exécution de tâches, et sans connaissance sur les requêtes futures. Par ailleurs, dans la plupart des systèmes actuels, les caractéristiques des tâches ne sont pas connues : il faudra donc les ordonnancer 'à l'aveuglette', sans connaissance de leurs temps d'exécution, parallélisme, dépendances entre les processus, etc.

Nous modélisons ce problème de la manière suivante. Des tâches arrivent au cours du temps et sont constituées d'un DAG (directed acyclic graph) de processus qui doivent être ordonnancés sur p processeurs. Nous utilisons le modèle introduit par Edmonds [1] pour modéliser des processus réalistes, dont la capacité à tirer parti des processeurs qui leurs sont alloués varie au cours du temps. On considère le modèle *en-ligne*, dans lequel les dates d'arrivée des tâches ne sont pas connues à l'avance, et *non-clairvoyant*, dans lequel les caractéristiques des tâches et des processus sont inconnues de l'ordonnanceur pendant l'exécution.

Les tâches. Une instance du problème est un ensemble $J_1, \ldots, J_n$ de n tâches. Chaque tâche J_i, relâchée au temps r_i est constituée d'un graphe orienté acyclique (DAG) $(\{J_{i1}, \ldots J_{im_i}, \prec\})$ de m_i processus ; le processus J_{ij} est activé dès que tous les processus J_{ik} tels que $J_{ik} \prec J_{ij}$ sont terminés. La tâche J_i est terminée lorsque tous ses processus le sont ; entre le moment r_i et la complétion de la tâche J_i, celle-ci sera dite *active*.

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

Les processus. Les processus sont ceux du modèle introduit par Edmonds [1]. Chaque processus J_{ij} passe par une série de *phases* : $J_{ij}^1, \ldots, J_{ij}^{q_{ij}}$, consistant chacune en une *quantité de travail* w_{ij}^k et un *profil d'accélération* $\Gamma_{ij}^k : \mathbb{R}^+ \to \mathbb{R}^+$. Au temps t, un processus J_{ij} qui est dans sa k-ème phase et recevant ρ processeurs progresse à vitesse $\Gamma_{ij}^k(\rho)$, c.-à-d. pendant un temps dt, il effectue un travail $dw = \Gamma_{ij}^k(\rho)dt$. Les différentes phases, quantités de travail et profils d'accélération sont inconnus de l'ordonnanceur.

Les fonctions Γ naturellement considérées sont supposées *croissantes* : $\Gamma(\rho) \leq \Gamma(\rho')$ si $\rho \leq \rho'$, c.-à-d. qu'augmenter le nombre de processeurs d'un processus ne peut qu'accélérer son exécution et *sous-linéaires* $\frac{\Gamma(\rho)}{\rho} \geq \frac{\Gamma(\rho')}{\rho'}$ si $\rho \leq \rho'$, c.-à-d. qu'un processus ne peut pas tirer parti plus efficacement des processeurs quand on augmente leur nombre. Dans la suite, on verra que dans le cadre de l'analyse, seulement deux profils d'accélération, extrêmes selon ces contraintes, seront à considérer : les phases *totalement parallèles*, notées PAR qui progressent à vitesse $\Gamma(\rho) = \rho$;et les phases *séquentielles*, notées SEQ qui progressent à vitesse constante quel que soit le nombre de processeurs qui leur est alloué : $\Gamma(\rho) = 1, \forall \rho \geq 0$.

Ordonnancement. Un ordonnancement S_p sur p processeurs est un ensemble de fonctions positives, constantes par morceaux $\rho_{ij} : t \to \rho_{ij}^t$, où ρ_{ij}^t est la quantité de processeurs allouée au processus J_{ij} au temps t, et tel que $\sum_{ij} \rho_{ij}^t \leq p$. Il est donc possible d'allouer une fraction arbitraire de processeur à un processus, ce qui, en pratique, est réalisé par 'time multiplexing'. Un ordonnancement est *valide* si tous les processus se terminent en un temps fini, et si, pour tous i, j, k, $\int_{r_{ij}^k}^{c_{ij}^k} \Gamma_{ij}^k(\rho_{ij}^t)dt = w_{ij}^k$, où r_{ij}^k est l'instant auquel la k-ème phase J_{ij}^k du processus J_{ij} est activée, et c_{ij}^k l'instant auquel elle termine.

Fonction objective. L'objectif est de minimiser le temps de flot (équivalent au temps moyen d'attente), défini par $\text{Flowtime}(S_p) = \sum_{i=1}^n c_i - r_i$. Un ordonnancement minimisant cette valeur sera noté OPT_p.

Un algorithme A est dit c-compétitif s'il calcule un ordonnancement S_p dont le temps de flot est au plus à un facteur c de l'optimum, $\text{Flowtime}(S_p) \leq c \cdot \text{Flowtime}(\text{OPT}_p)$. Edmonds [1] a démontré qu'aucun algorithme en-ligne non-clairvoyant ne peut avoir un facteur de compétitivité constant. Pour palier ce problème et pouvoir comparer les algorithmes entre eux, comme [1, 2], nous utilisons la technique d'augmentation de ressources, qui consiste à favoriser l'algorithme que l'on étudie en lui attribuant plus de processeurs qu'à l'optimum. Ainsi, un algorithme A est dit *s-speed c-compétitif* s'il calcule un ordonnancement S_{sp} sur sp processeurs dont le temps de flot est au plus à un facteur c de l'optimum sur p processeurs : $\text{Flowtime}(S_{sp}) \leq c \cdot \text{Flowtime}(\text{OPT}_p)$. Dans la suite, sans perte de généralité, on prendra $p = 1$.

L'algorithme $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$. Nous nous intéressons à une famille particulière d'ordonnanceurs, qui sont la composition d'un ordonnanceur de tâches et d'un ordonnanceur de processus. *L'ordonnanceur de tâches* attribue des processeurs aux tâches actives qui sont ensuite répartis entre les processus actifs de la tâche par *l'ordonnanceur de processus*. L'algorithme $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$ est la composition de $\mathbf{LAPS}_\beta$ comme ordonnanceur de tâches et d'**EQUI** comme ordonnanceur de processus. L'algorithme $\mathbf{LAPS}_\beta$,

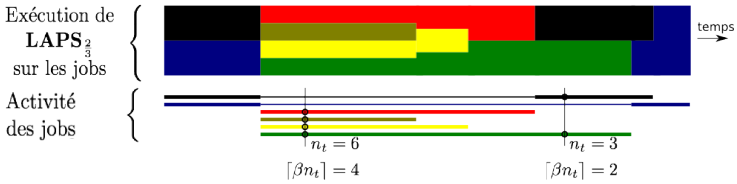


FIGURE 1: Exécution de l'algorithme $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$. Les structures des tâches ne sont pas représentées. Les processus sont ordonnancés par **EQUI**.

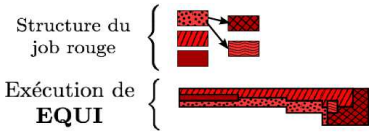


FIGURE 2: Exécution d'**EQUI**, sur le job rouge de l'exemple ci-contre.

représenté sur la figure 1 attribue équitablement les processeurs à la fraction β des tâches arrivées le plus récemment : soient $J_1, \ldots J_{n_t}$ les n_t tâches actives à l'instant t, classées par date d'arrivée croissante ; chacune des $\lceil \beta n_t \rceil$ tâches J_i, $i \geq (n_t - \lceil \beta n_t \rceil)$, reçoit $\frac{1}{\lceil \beta n_t \rceil}$ processeurs. Ces processeurs reçus sont alors partagés équitablement par **EQUI** entre les processus actifs de chaque tâche (cf. figure 2) : pour tout $i \geq (n_t - \lceil \beta n_t \rceil)$, chaque processus J_{ij} reçoit alors $\frac{1}{n_t^i \lceil \beta n_t \rceil}$, où n_t^i est le nombre de processus actifs de la tâche J_i.

Pour tout $s \geq 1 + \beta + \varepsilon$, Edmonds et Pruhs [2] démontrent que l'algorithme $\mathbf{LAPS}_\beta$ est s-speed $\frac{4s}{\beta\varepsilon}$-

compétitif en l'absence de dépendances ; [3], démontre que $\mathbf{LAPS}_\beta$ est en fait s-speed $\frac{2s}{\beta\varepsilon}$-compétitif.

2 Analyse de $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$

Dans cette section, nous montrons que l'algorithme $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$ est compétitif sur s fois plus de processeurs que l'optimum, avec $s \geq 1+\beta+\varepsilon$. L'introduction de dépendances ne détériore donc la qualité de service que d'un facteur constant, et la charge qu'il est possible de soutenir est la même en présence de dépendances. Pour l'analyse d'un ordonnanceur A_s, de même que dans [3], on montre que les instances les plus difficiles sont constituées uniquement de travail PAR et SEQ, et que A_s est toujours en retard par rapport à OPT sur le travail PAR. La preuve nécessite un peu plus de précautions que dans [3], puisque les processus ne sont plus activés aux mêmes moment dans OPT et dans A_s, mais on montre par induction sur le DAG que les phases PAR sont en fait activées plus tard dans A_s que dans OPT.

En l'absence de dépendances, [2, 3] réussissent à borner Flowtime($\mathbf{LAPS}_\beta$) par la quantité totale de travail SEQ des tâches, $\int_0^\infty \ell_t dt$ où ℓ_t est le nombre de tâches dans une phase SEQ au temps t. Cette quantité est clairement un minorant de Flowtime(OPT), puisque OPT ne peut pas accélérer les phases SEQ.

En présence de dépendances, pour définir le paramètre ℓ_t, nous transformons l'ordonnancement renvoyé par $\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$ sans modifier son temps de flot, de façon à n'avoir, à chaque instant, qu'un seul type de phases ordonnancées dans chaque tâche ; ℓ_t sera alors le nombre de tâches dont le type de phases ordonnancées au temps t est SEQ. Grâce à ce paramètre, on obtient, pour Flowtime($\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$), la même borne que [3], Flowtime($\mathbf{LAPS}_\beta \circ \mathbf{EQUI}$) $\leq \frac{2s}{\beta\varepsilon} \int_0^\infty \ell_t dt$.

La principale difficulté est alors de relier OPT à $\int_0^\infty \ell_t dt$. En effet, comme l'illustre la figure 3, les choix de l'ordonnanceur de processus peuvent mener à dérouler le DAG de façon à étaler beaucoup plus le travail SEQ que OPT, et $\int_0^\infty \ell_t dt$ ne sera plus un minorant trivial de OPT.

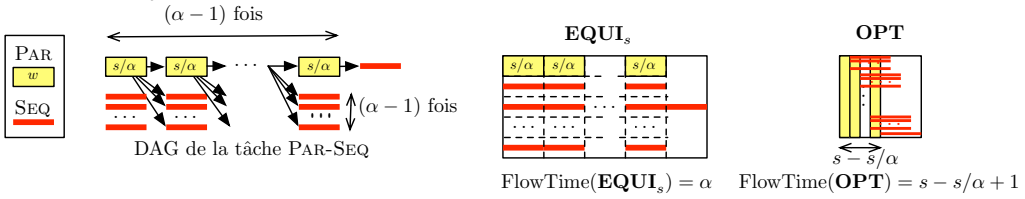


FIGURE 3: Dans $\mathbf{EQUI}_s$, les phases SEQ sont étalées d'un facteur $\frac{\alpha}{s}$ par rapport à OPT

Pour faire cette minoration, on introduit naturellement la notion de *gâchis*, correspondant à la proportion de processeurs gâchés sur les phases SEQ :

Définition (Gâchis)**.** Le *gâchis* d'un ordonnanceur de processus A est l'intégrale sur le temps de la proportion de processeurs attribués à des processus dans une phase SEQ (cf. figure 4) : soit w_i^t la quantité de processeurs qui ne sont pas attribués à des phases PAR au temps t dans une tâche J_i : $w_i^t = \rho_i^t - \pi_i^t$, où $\pi_i^t = \sum_{j \in \mathrm{PAR}_i^t} \rho_{ij}^t$, $\mathrm{PAR}_i^t = \{j : J_{ij}$ est actif et dans une phase PAR au temps $t\}$.

On note gâchis$(A, \rho, J_i) = \int_0^\infty \frac{w_i^t}{\rho_i^t} dt$, où ρ_i^t la quantité de processeurs attribués à A pour J_i par l'ordonnanceur de tâches.

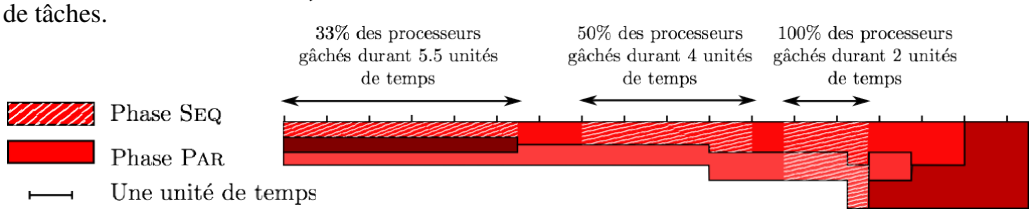
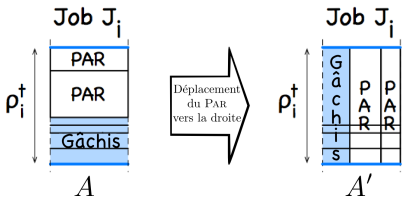


FIGURE 4: Illustration du gâchis sur une exécution d'$\mathbf{EQUI}$. Ici, gâchis $= 5.5 \cdot 1/3 + 2 \cdot 1/2 + 2 \cdot 1 = 5.833$

Pour se ramener à l'analyse sans dépendance, on construit, à partir d'un ordonnancement $A(J_i)$, un nouvel ordonnancement $A(J_i)'$ de même gâchis, et tel qu'à tout instant, exactement un processus est exécuté. L'idée est de : découper le temps en intervalles suffisamment courts pendant lesquels aucun processus ne change de phase et où l'allocation de processeurs ρ ne varie pas ; puis dans chacun de ces intervalles,



ordonnancer le travail SEQ au début, c.-à-d. déplacer le PAR vers la droite, comme représenté sur la figure ci-contre.

On peut alors voir une tâche J_i dans l'ordonnancement ainsi obtenu comme une phase de travail parallèle total égal à la somme des travaux parallèles des processus de J_i et de travail séquentiel total égal à *gâchis*(A,ρ,J_i). Il s'agit à présent de relier le gâchis au travail SEQ dans chaque tâche et plus précisément à la plus longue chaîne de travail SEQ d'une tâche J_i, notée $\text{SEQ}(J_i)$, qui est une borne inférieure sur le temps d'activité de J_i dans OPT. Ceci nous permettra alors de relier $\int_0^\infty \ell_t dt$ et Flowtime(OPT). Un algorithme sera dit α-scatterer si le gâchis ne dépasse pas $\text{SEQ}(J_i)$ de plus d'un facteur α.

Définition (α-scatterer)**.** On note $\text{SEQ}(J_{ij}) = \sum_{k:J_{ij}^k \text{ est une phase SEQ}} w_{ij}^k$ la quantité de travail SEQ d'un processus J_{ij}, et $\text{SEQ}(J_i)$ la plus grande quantité de SEQ le long d'une chaîne de J_i : $\text{SEQ}(J_i) = \max_{J_{ii_1} \prec J_{ii_2} \prec \ldots \prec J_{ii_q}} \sum_j \text{SEQ}(J_{ii_j})$. Un α-*scatterer* est un ordonnanceur de processus A tel que pour toute allocation de processeur $\rho : \mathbb{R}^+ \to \mathbb{R}^+$ et pour toute tâche J_i, $\text{gâchis}(A,\rho,J_i) \leq \alpha \text{SEQ}(J_i)$.

Deux processus J_{ij} et J_{ik} d'une tâche J_i sont dits indépendants si $J_{ij} \not\prec^* J_{ik}$ et $J_{ik} \not\prec^* J_{ij}$, où $\not\prec^*$ est la clôture transitive de $\prec$. Une anti-chaîne d'une tâche J_i est un ensemble de processus de J_i deux à deux indépendants.

Proposition 1. *L'ordonnanceur de processus* **EQUI** *est un* $\frac{k+1}{2}$*-scatterer, pour tout majorant k de la longueur maximale d'une anti-chaîne d'une tâche.*

Démonstration. Soit J_i une tâche ayant une anti-chaîne de taille k, et ρ_i une allocation de processeurs pour J_i. Soit ν_t le nombre total de processus de J_i actifs au temps t et λ_t le nombre de processus actifs dans une phase SEQ. Par définition d'**EQUI**, les processus dans une phase SEQ reçoivent $\frac{\lambda_t}{\nu_t}\rho_i^t$ processeurs au temps t, d'où $\text{gâchis}(\mathbf{EQUI},\rho_i,J_i) = \int_0^\infty (\frac{\lambda_t}{\nu_t} \cdot \rho_i^t) \cdot \frac{1}{\rho_i^t} dt = \int_0^\infty \frac{\lambda_t}{\nu_t} dt$. D'après le thèorème de Dilworth, on peut partitionner les processus de J_i en au plus k chaînes disjointes, $\xi_1, \ldots \xi_k$. Comme **EQUI** ordonnance chaque processus dès qu'il est actif, on peut supposer sans perte de généralité que ξ_1 contient un processus actif à tout moment jusqu'à ce que J_i termine. Soit $\mathbf{1}_{\xi_j}(t) = 1$ s'il y a un processus actif de ξ_j dans une phase SEQ au temps t, et $\mathbf{1}_{\xi_j}(t) = 0$ sinon. Comme les k chaînes couvrent tous les processus, on peut récrire $\text{gâchis}(\mathbf{EQUI},\rho,J_i) = \int_0^\infty \frac{\sum_{j=1}^k \mathbf{1}_{\xi_j}(t)}{\nu(t)} dt = \sum_{j=1}^k \int_0^\infty \frac{\mathbf{1}_{\xi_j}(t)}{\nu(t)} dt \leq \int_0^\infty \mathbf{1}_{\xi_1}(t) dt + \sum_{j=2}^k \int_0^\infty \frac{\mathbf{1}_{\xi_j}(t)}{2} dt$ puisque toute chaîne ξ_j, $j \geqslant 2$ est exécutée avec au moins un processus de ξ_1. On a alors $\text{gâchis}(\mathbf{EQUI},\rho,J_i) \leq \text{SEQ}(\xi_1) + \frac{k-1}{2} \max\{\text{SEQ}(\xi_2), \ldots \text{SEQ}(\xi_k)\} \leq \frac{k+1}{2} \text{SEQ}(J_i)$ par définition de $\text{SEQ}(J_i)$. □

Théorème 1. *Pour tout* α*-scatterer A, l'algorithme* $\mathbf{LAPS}_\beta \circ A$ *est s-speed* $\frac{2s\alpha}{\beta\varepsilon}$*-compétitif pour tout* $s \geq 1+\beta+\varepsilon$.

Démonstration. Considérons l'ordonnancement obtenu après avoir effectué le déplacement du PAR vers la droite pour toutes les tâches. $\mathbf{LAPS}_\beta \circ A$ reste en retard sur OPT, et le travail SEQ est au plus augmenté d'un facteur α. On conclut en faisant les mêmes calculs que [3], et en minorant $\alpha \cdot \text{Flowtime(OPT)}$ par $\int_0^\infty \ell_t dt$. □

Références

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

[2] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *Proc. of ACM/SIAM Symp. On Discrete Algorithms (SODA)*, pages 685–692, 2009.

[3] J. Robert and N. Schabanel. Ordonnancement non-clairvoyant : petites simplifications et améliorations de l'analyse de la famille d'algorithmes $\mathbf{LAPS}_\beta$. Soumis à Algotel 2009.

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