Délégation de lots de tâches pour la réduction de la durée moyenne de réalisation

Ellie Beauprez, Anne-Cécile Caron, Maxime Morge, Jean-Christophe Routier

Ellie Beauprez, Anne-Cécile Caron,
Maxime Morge, Jean-Christophe Routier

Délégation de lots de tâches pour la réduction de la durée moyenne de réalisation

Volume 4, nº 2 (2023), p. 193-221.

DOI not yet assigned

La Revue Ouverte d'Intelligence Artificielle est membre du Centre Mersenne pour l'édition scientifique ouverte
www.centre-mersenne.org
e-ISSN : 2967-9672

Revue Ouverte d'Intelligence Artificielle
Volume 4, n° 2, 2023, 193-221

Délégation de lots de tâches pour la réduction de la durée moyenne de réalisation

Ellie Beauprez[a], Anne-Cécile Caron[a], Maxime Morge[a], Jean-Christophe Routier[a]

[a] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
E-mail : Ellie.Beauprez@univ-lille.fr, Anne-Cecile.Caron@univ-lille.fr, Maxime.Morge@univ-lille.fr, Jean-Christophe.Routier@univ-lille.fr.

Résumé. — Nous étudions le problème de la réallocation de tâches pour l'équilibrage de charge dans les modèles distribués de traitement de données massives. Nous proposons une stratégie qui repose sur des agents coopératifs pour optimiser le réordonnancement de tâches dans un ensemble de jobs devant être exécutés le plus tôt possible. Elle permet aux agents de déterminer localement les prochaines tâches à exécuter, à déléguer, voire à échanger grâce à leurs connaissances, leurs croyances et leur modèle des pairs. La nouveauté réside dans la capacité des agents à identifier les opportunités et les agents limitants pour réallouer efficacement des lots de tâches à travers des négociations bilatérales concurrentes. La stratégie mise en œuvre par les agents permet de garantir une amélioration continue du délai de réalisation. Nos expérimentations montrent que la durée moyenne de réalisation atteinte par notre stratégie est meilleure que celle obtenue avec une résolution DCOP et reste proche de celle obtenue avec une heuristique classique, avec dans tous les cas un temps de réordonnancement significativement réduit.

Mots-clés. — Système multi-agents, résolution collective de problèmes, négociation multi-agents.

1. Introduction

La problématique de l'affectation efficace de tâches parmi des entités exécutantes est commune à de nombreuses applications réelles pour la logistique [16, 18], la robotique collective [9, 26], le calcul distribué [15, 24], ou plus récemment le traitement de données massives [1]. En particulier, les sciences des données, qui exploitent de larges volumes de données sur lesquelles des calculs sont effectués en parallèle par différents nœuds, mettent à l'épreuve l'informatique distribuée en ce qui concerne l'allocation de tâches et l'équilibrage de charge. Cet article traite d'une classe d'applications pratiques où : (a) des jobs (c'est-à-dire des ensembles de tâches) concurrents doivent êtres exécutés le plus tôt possible, et (b) les ressources (c'est-à-dire les données) requises sont distribuées. Nous considérons particulièrement le modèle de traitement le plus répandu pour traiter des données massives sur une grappe de serveurs, c'est-à-dire le patron de conception MapReduce [28]. Les jobs y sont composés d'un ensemble de tâches

exécutées par les différents nœuds où sont réparties les ressources. Comme plusieurs ressources sont requises pour réaliser une tâche sur un nœud, son exécution nécessite de récupérer des ressources disponibles sur d'autres nœuds, ce qui induit un surcoût.

De nombreux travaux adoptent le paradigme multi-agents pour aborder le problème de la réallocation de tâches et de l'équilibrage de charge dans les systèmes distribués [15]. L'approche centrée individus permet la distribution d'heuristiques pour des problèmes impraticables à cause de la combinatoire des ordonnancements pour permettre le passage à l'échelle. De plus, intrinsèquement réactives, les méthodes multi-agents de réaffectation s'adaptent aux estimations inexactes des temps d'exécution et aux perturbations (consommation/libération de tâches, ralentissement des exécutants, etc.) La plupart de ces travaux adoptent l'approche orientée marché en modélisant le problème comme un jeu non-coopératif [9, 27], voire en s'appuyant sur des méthodes d'apprentissage qui nécessitent un historique [24, 26]. Par contraste, nous supposons comme [1] que : (a) les agents sont coopératifs, c'est-à-dire ils ont une perception locale et partielle de l'allocation, mais ils partagent le même objectif, et (b) aucun modèle préalable n'existe, ni des données, ni de l'environnement, de par la classe d'applications pratiques visées. Nous allons ici au-delà en considérant plusieurs jobs composés d'ensembles de tâches, chacune pouvant être exécutée par un seul des nœuds exécutants, tous compétents. Chaque agent, représentant un nœud, souhaite minimiser la durée moyenne de réalisation des jobs, c'est-à-dire le *flowtime* moyen. La principale difficulté réside dans la formulation de comportements individuels joués de manière asynchrone par les exécutants qui doivent aboutir à l'émergence d'affectations faisables qui combinent les objectifs des commanditaires.

Nous proposons des stratégies qui décident quelle réallocation bilatérale est suggérée ou acceptée. Elles reposent sur un modèle des pairs et déterminent le comportement de l'agent à chaque point de choix dans le protocole de négociation. La stratégie d'offre sélectionne une délégation potentielle, c'est-à-dire un lot de tâches et un répondant. La stratégie d'acceptation détermine si l'agent accepte ou refuse toute ou partie de cette délégation.

Nos contributions sont les suivantes.

(1) Nous formalisons le problème d'allocation multi-agents de jobs composés de tâches situées dont les coûts diffèrent selon la localisation des ressources.
(2) Nous proposons une stratégie qui identifie en continu les agents limitants et les opportunités au sein de répartitions déséquilibrées pour déclencher des négociations concurrentes et bilatérales afin de déléguer voire d'échanger des tâches.
(3) Nous avons conduit des expériences approfondies qui montrent que notre méthode atteint une durée de réalisation proche de celle obtenue par l'heuristique classique et réduit significativement le temps de réordonnancement.

Cet article est une version étendue de [3] où :

(1) la notion de délégation a été généralisée sous la forme de réallocation bilatérale ce qui permet d'envisager des délégations n-aires et des échanges ;

(2) la rationalité des réallocations a été redéfinie, ce qui a permis de réduire non seulement la durée moyenne de réalisation atteinte par notre stratégie mais également le temps de réordonnancement ;
(3) nous comparons les performances de notre stratégie à celles des techniques d'optimisation distribuée sous contraintes (DCOP).

Après un aperçu des travaux connexes dans la section 2, nous formalisons le problème d'allocation multi-agents de jobs composés de tâches (*cf.* section 3). La section 4 décrit les opérations de consommation/délégation et le processus de négociation. La section 5 précise comment les agents choisissent quelles tâches négocier et avec qui. Notre évaluation expérimentale est décrite dans la section 6. La section 7 résume notre contribution et présente nos perspectives.

2. Travaux connexes

	Ressources	**Tâches**	**Exécutants**	**Objectif**	**Dynamique**	**Décentralisé**	**Technique/ Modèle**
(Kuhn-Munkres, 1955) [16]	—	n mono-exécutant	R_n mono-tâche	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✗	✗	LP
(SPT, 1967) [10]	—	n mono-exécutant	P_m multi-tâches	$\mathrm{C}(\overrightarrow{\mathrm{A}})$	✗	✗	Heuristique
(Bruno *et al.*, 1974) [7]	—	n mono-exécutant	R_m multi-tâches	$\mathrm{C}(\overrightarrow{\mathrm{A}})$	✗	✗	LP
(Shehory et Krause, 1998) [25]	—	n multi-exécutant	R_m mono-tâche	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	Coalition
(GPGP, 2004) [17]	consommables ou pas	n mono-exécutant	R_m multi-tâches	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	Équipe
(Turner *et al.*, 2018) [26]	limitées	n mono-exécutant	R_m en ligne multi-tâches	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	CBBA + classification
(Li *et al.*, 2014) [18]	—	n mono-exécutant	P_m multi-tâches	$\mathrm{C}(\overrightarrow{\mathrm{A}})$ $\oplus\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	DCOP
(Schaerf *et al.*, 1995) [24]	—	n mono-exécutant	P_m multi-tâches	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	MARL
(MASTA, 2021) [1]	transférables duplicables	n mono-exécutant	R_m multi-tâches	$\mathrm{C}_{\max}(\overrightarrow{\mathrm{A}})$	✓	✓	Équipe
(SMASTA+, 2021) [4]	transférables duplicables	n mono-exécutant	R_m multi-tâches	$\mathrm{C}(\overrightarrow{\mathrm{A}})$	✓	✓	Équipe

Table 2.1 – Grille d'analyse des méthodes d'affectation (haut) ou de réaffectation (bas)

Le tableau 2.1 synthétise l'ensemble des travaux évoqués ici selon notre grille d'analyse. Cette grille fait référence à un état de l'art plus large que nous avons publié [2]. La partie gauche du tableau discerne les différents problèmes abordés : leurs ingrédients (ressources, tâches, exécutants) et les objectifs visés. La partie droite révèle les caractéristiques de ces méthodes ainsi que les techniques et modèles sous-jacents. La partie supérieure du tableau contient les méthodes classiques d'affectation alors que la partie inférieure présente les méthodes de réaffectation dynamique et continue.

La théorie de l'ordonnancement [8] propose des méthodes hors-ligne pour résoudre différents problèmes d'affectation efficace de tâches parmi des entités exécutantes. Par

exemple, l'algorithme polynomial de Kuhn-Munkres, appelé méthode hongroise, minimise le coût total de l'affectation (noté $W(\overrightarrow{A})$) de n tâches à n exécutants [16]. La règle du « plus court processus en premier » – *shortest processing time first* (SPT) – est une méthode très simple qui minimise le débit de réalisation pour un problème avec un seul exécutant multi-tâches et n tâches mono-exécutant [10]. Ce résultat se généralise au problème avec m exécutants multi-tâches si le coût d'une tâche est identique d'un exécutant à l'autre (noté P_m). Le problème d'ordonnancement qui consiste à minimiser la durée totale de réalisation (notée $C(\overrightarrow{A})$) avec m exécutants multi-tâches et n tâches mono-exécutant dont les coûts dépendent de l'entité exécutante (noté R_m) peut être formalisé par programmation linéaire – *Linear Programming* (LP). Ce problème se réduit à un problème d'appariement pondéré dans un graphe biparti avec n tâches et $n \times m$ positions. Ce problème est polynomial [14]. Reposant sur l'algorithme de Ford-Fulkerson, la complexité de l'algorithme décrit par [7] est $O(\max(mn^2, n^3))$. Ces approches ne sont pas toujours adaptées à la réallocation de tâches dans des systèmes distribués où décentralisation et adaptativité sont nécessaires. En effet, un contrôle global constitue un goulot d'étranglement en matière de performance, car il doit en permanence collecter des informations sur l'état du système. À l'opposé, nos agents prennent des décisions locales sur une allocation existante dans le but d'améliorer l'équilibrage de charges. De plus, les problèmes d'ordonnancement classiques sont statiques. L'estimation inexacte du temps d'exécution des tâches, aggravée par des perturbations (consommation de tâches, libération de jobs, ralentissement des nœuds, etc.) peut nécessiter d'importantes modifications de l'allocation existante pour qu'elle reste optimale. Qui plus est, les agents peuvent agir dans des environnements dynamiques qui évoluent au cours du temps.

Le paradigme multi-agents est particulièrement approprié pour la conception et l'implémentation de mécanismes distribués et adaptatifs de réaffectation de tâches-exécutants [15]. Les modèles existants se distinguent de par la nature des tâches et des agents, qu'ils représentent les exécutants ou les commanditaires des tâches. Les modèles de formation d'une coalition se justifient si la réalisation d'une tâche nécessite plus d'un exécutant ou si son coût diminue avec le nombre d'exécutants affectés. Par exemple, Shehory et Kraus proposent des algorithmes décentralisés gloutons et *anytime* pour l'assignation de tâches multi-exécutants et soumises à des contraintes de précédence, à des exécutants dont les capacités/efficacités sont hétérogènes [25]. Une équipe vise à maximiser une fonction objectif globale plutôt que la satisfaction individuelle des membres, comme pour une coalition, mais les tâches sont mono-exécutant. En particulier, Lesser et al. proposent un modèle hiérarchique de représentation des tâches pour la coordination qui est indépendant du domaine : affecter les ressources, assigner les tâches et les ordonnancer [17]. L'objectif premier de *Generalized Partial Global Planning* (GPGP) est de maximiser l'utilité combinée de l'ensemble des agents par la réalisation de ses objectifs de plus haut niveau. GPGP est un modèle de coordination basée sur la planification des activités. Cette approche suppose que l'effort nécessaire à la coordination (raisonnement et communication) est négligeable par rapport aux temps d'exécution des tâches à coordonner. S'inspirant de théories économiques, la « programmation orientée marché » aborde les problèmes de planification

distribuée à travers la recherche d'un équilibre pour un jeu non-coopératif [27]. Les agents délèguent des (lots de) tâches voire les échangent. Contrairement à une équipe, une place de marché suppose que les contraintes et les objectifs sont complètement distribués. Parmi les méthodes orientées marché, on distingue trois familles.

DCOP. Les problèmes de réaffectation peuvent être représentés sous la forme d'un problème d'optimisation sous contraintes distribué – *Distributed Constraint Optimization Problems* (DCOP). De nombreuses méthodes ont été développées pour la recherche d'une solution optimale à un DCOP qui est un problème NP-difficile (voir [12] pour une synthèse récente). La principale difficulté dans la mise en œuvre de ces méthodes pour la réaffectation de tâches réside dans la représentation d'un problème réaliste sous la forme d'un DCOP, voire de plusieurs sous-problèmes COP, car elle nécessite une expertise de la méthode de résolution (*e.g.* [18]).

CBBA. L'algorithme à base de consensus (CBBA – *Consensus Based Bundle Algorithm*) [9] est une méthode multi-agents d'affectation en deux phases qui consiste à : (a) sélectionner les tâches à négocier ; (b) déterminer l'agent qui remporte ces négociations. Dans la continuité, Turner et al. étudient l'affectation en continu de tâches à une flotte de robots pour maximiser le débit de réalisation avec des ressources en carburant limitées [26]. Grâce à l'apprentissage automatique supervisé à partir des exécutions précédentes, les robots choisissent dynamiquement et de manière décentralisée la meilleure heuristique de sélection de tâche.

MARL. Les problèmes de réaffectation peuvent également être modélisés via des processus de décision markoviens [6], en particulier des processus de décision markoviens partiellement observables décentralisés – *Decentralized Partially Observed Markov Decision Process* (Dec-POMDP). L'optimisation d'un Dec-POMDP à horizon fini est un problème NEXPTIME. Les méthodes de résolution approchée ne peuvent être appliquées que sur de très petites instances de problèmes, elles ne passent pas à l'échelle. Au-delà de ces méthodes de planification hors-ligne, l'apprentissage multi-agents par renforcement (MARL-*Multi-Agent Reinforcement Learning*) nécessite une connaissance parfaite de l'environnement et requiert une phase d'apprentissage [24].

À l'inverse, nous ne considérons aucun modèle préalable, ni des données, ni de l'environnement, car cela ne serait pas pertinent pour la classe d'applications pratiques qui nous concerne. Le volume de données rend son pré-traitement trop coûteux et la variabilité des données rend rapidement le pré-traitement obsolète. De la même manière, Baert *et al.* visent dans [1] un objectif égalitaire qui est la minimisation du temps nécessaire à la réalisation de l'ensemble des tâches (noté $C_{max}(\overrightarrow{A})$). Nous considérons ici le problème de la coordination des décisions entre agents pour trouver une solution globalement optimale pour des fonctions multi-objectifs. Les agents tentent de minimiser la durée moyenne de réalisation de plusieurs jobs concurrents, chacun constitué de plusieurs tâches. Les tâches considérées sont mono-exécutants et les ressources qu'elles requièrent sont transférables et non consommables. Les agents exécutants sont multi-tâches et tous compétents.

Alors que nos travaux précédents ne considèrent que des délégations d'une seule tâche à la fois, nous généralisons ici notre cadre formel pour considérer n'importe quelle réallocation bilatérale (échange ou délégation de plusieurs tâches) afin de réduire non seulement la durée moyenne de réalisation mais également le temps de réordonnancement.

Nous redéfinissons ici la rationalité des réallocations. Précédemment, ce critère était défini à partir du *flowtime* local, c'est-à-dire la durée de réalisation pour les deux agents impliqués dans la transaction. Comme ce critère seul ne permet pas de garantir la terminaison de l'algorithme multi-agents de réallocation, il était combiné avec le *makespan*, le temps nécessaire à la réalisation de l'ensemble des jobs. Dans cet article, une réallocation bilatérale est rationnelle si elle réduit le *flowtime* global du système. Ce critère s'avère suffisant pour garantir la convergence du processus de réallocation. De plus, il permet de réduire le temps de réordonnancement et la durée moyenne de réalisation atteinte par notre stratégie.

Nous comparons quantitativement les performances de notre stratégie à celles des techniques d'optimisation distribuée sous contraintes (DCOP).

3. Tâches situées

Nous formalisons ici le problème d'allocation multi-agents des jobs concurrents composés de tâches situées.

Un job est un ensemble de tâches indépendantes, non divisibles et non-préemptives. L'exécution de chaque tâche nécessite l'accès à des ressources distribuées sur les nœuds du système. Nous considérons les ressources transférables et non consommables.

Définition 3.1 (Système distribué). — *Un système distribué est un triplet* $\mathscr{D} = \langle \mathscr{N}, \mathscr{E}, \mathscr{R} \rangle$ *où :*

- $\mathscr{N} = \{\nu_1, \ldots, \nu_m\}$ *est un ensemble de nœuds ;*
- $\mathscr{E}$ *est une relation d'accointance, i.e. une relation binaire et symétrique sur* $\mathscr{N}$ *;*
- $\mathscr{R} = \{\rho_1, \ldots, \rho_k\}$ *est un ensemble de ressources de tailles* $|\rho_i|$*. La localisation des ressources, éventuellement répliquées, est déterminée par la fonction :*

$$l : \mathscr{R} \to 2^{\mathscr{N}} \tag{3.1}$$

Pour plus de simplicité, nous faisons l'hypothèse ici qu'il y a exactement un agent par nœud et que toutes les ressources sont accessibles pour tous les agents même celles sur les nœuds distants car la relation d'accointance est totale.

L'exécution d'un job (sans date butoir) consiste à exécuter un ensemble de tâches indépendantes nécessitant des ressources pour produire un résultat.

Définition 3.2 (Job/Tâche). — *Soient* $\mathscr{D}$ *un système distribué et Res l'espace des résultats. On considère un ensemble de* ℓ *jobs* $\mathscr{J} = \{J_1, \ldots, J_\ell\}$*. Chaque job* J_i

est un ensemble de k_i tâches $J_i = \{\tau_1, \ldots, \tau_{k_i}\}$ où chaque tâche τ est une fonction $\tau : 2^{\mathscr{R}} \mapsto Res$.

On note $\mathscr{T} = \cup_{1 \leqslant i \leqslant \ell} \mathrm{J}_i$ l'ensemble des n tâches sous-jacentes à $\mathscr{J}$ et $\mathscr{R}_\tau \subseteq \mathscr{R}$ l'ensemble des ressources requises pour la tâche τ. Par souci de concision, on note $\mathrm{job}(\tau)$ le job contenant la tâche τ. Nous faisons l'hypothèse que le nombre de jobs est négligeable par rapport au nombre de tâches, $|\mathscr{J}| \ll |\mathscr{T}|$.

Le coût d'une tâche est une estimation de son temps d'exécution par un nœud.

Propriété 3.3 (Coût). — *Soient $\mathscr{D}$ un système distribué et $\mathscr{T}$ un ensemble de tâches. La fonction de coût $c : \mathscr{T} \times \mathscr{N} \mapsto \mathbb{R}_+^*$ est telle que :*

$$c(\tau, \nu_i) \leqslant c(\tau, \nu_j) \Leftrightarrow \Sigma_{\rho \in \mathscr{R}_\tau, \nu_i \in l(\rho)} |\rho| > \Sigma_{\rho \in \mathscr{R}_\tau, \nu_j \in l(\rho)} |\rho| \tag{3.2}$$

Comme la collecte de ressources distantes représente un surcoût, une tâche est moins coûteuse si les ressources nécessaires sont « plus locales » (*cf.* section 6). La fonction de coût peut être étendue à un ensemble de tâches :

$$\forall \mathrm{T} \subseteq \mathscr{T},\ \mathrm{c}(\mathrm{T}, \nu_i) = \Sigma_{\tau \in \mathrm{T}} \mathrm{c}(\tau, \nu_i) \tag{3.3}$$

En substance, nous considérons le problème d'allocation multi-agents de jobs composés de tâches situées.

Définition 3.4 (MASTA+). — *Un problème d'allocation multi-agents de jobs est un quadruplet MASTA+ $= \langle \mathscr{D}, \mathscr{T}, \mathscr{J}, c \rangle$ où :*

- *$\mathscr{D}$ est un système distribué de m nœuds ;*
- *$\mathscr{T} = \{\tau_1, \ldots, \tau_n\}$ est un ensemble de n tâches ;*
- *$\mathscr{J} = \{J_1, \ldots, J_\ell\}$ est un partitionnement des tâches en ℓ jobs ;*
- *$c : \mathscr{T} \times \mathscr{N} \mapsto \mathbb{R}_+^*$ est la fonction de coût.*

Une allocation de tâches est une répartition des tâches dans des lots ordonnés.

Définition 3.5 (Allocation). — *Une allocation pour un problème MASTA+ est un vecteur de m lots de tâches ordonnées $\overrightarrow{A} = ((B_1, \prec_1), \ldots, (B_m, \prec_m))$ où chaque lot $(B_i, \prec_i)$ est l'ensemble des tâches ($B_i \subseteq \mathscr{T}$) affectées au nœud ν_i associé à un ordre total strict ($\prec_i \subseteq \mathscr{T} \times \mathscr{T}$). $\tau_j \prec_i \tau_k$ signifie que si $\tau_j, \tau_k \in B_i$ alors τ_j est exécutée avant τ_k par ν_i. L'allocation $\overrightarrow{A}$ vérifie :*

$$\forall \tau \in \mathscr{T},\ \exists \nu_i \in \mathscr{N},\ \tau \in B_i \tag{3.4}$$

$$\forall \nu_i \in \mathscr{N}, \forall \nu_j \in \mathscr{N} \setminus \{\nu_i\},\ B_i \cap B_j = \emptyset \tag{3.5}$$

Toutes les tâches sont allouées (équation (3.4)) et chacune n'est allouée qu'à un seul nœud (équation (3.5)). Par souci de concision, on note :

- $\overrightarrow{\mathrm{B}}_i = (\mathrm{B}_i, \prec_i)$, le lot trié de ν_i ;
- $\min_{\prec_i} \mathrm{B}_i$, la prochaine tâche à exécuter par ν_i :

- $\text{jobs}(\text{B}_i)$, l'ensemble des jobs affectés à ν_i, *i.e.* les jobs ayant au moins une tâche dans B_i ;
- $\nu(\tau, \overrightarrow{\text{A}})$, le nœud chargé de τ dans $\overrightarrow{\text{A}}$;
- $\text{w}_i(\overrightarrow{\text{A}}) = \Sigma_{\tau \in \text{B}_i} \text{c}(\tau, \nu_i)$, la charge de travail du nœud ν_i pour l'allocation $\overrightarrow{\text{A}}$.

Par abus de notation, une allocation réduite à un singleton, $\overrightarrow{\text{A}} = (\overrightarrow{\text{B}}_i)$, est notée $\overrightarrow{\text{A}} = \overrightarrow{\text{B}}_i$.

Comme on suppose que les nœuds sont toujours actifs, la durée de réalisation d'une tâche (*completion time*) correspond au temps d'attente avant que la tâche soit entamée plus l'estimation de son temps d'exécution :

$$\text{C}_\tau(\overrightarrow{\text{A}}) = \text{t}(\tau, \nu(\tau, \overrightarrow{\text{A}})) + \text{c}(\tau, \nu(\tau, \overrightarrow{\text{A}})) \quad (3.6)$$
$$\text{avec } \text{t}(\tau, \nu_i) = \Sigma_{\tau' \in \text{B}_i \mid \tau' \prec_i \tau} \text{c}(\tau', \nu_i)$$

Contrairement aux coûts, les durées de réalisation dépendent de l'ordre d'exécution.

Pour évaluer la qualité d'une allocation de tâches, nous considérons le *flowtime* moyen, qui mesure le temps moyen écoulé entre la date de libération des jobs (on suppose ici que tous les jobs sont libérés à la même date $t = 0$) et leur date d'achèvement, et le *makespan* qui est le temps nécessaire à la réalisation de l'ensemble des jobs.

Définition 3.6 (*Flowtime/Makespan*). — *Soient MASTA+ un problème d'allocation de tâches et $\overrightarrow{A}$ une allocation. On définit :*

- *la durée de réalisation de $J \in \mathcal{J}$ pour $\overrightarrow{A}$,*
$$C_J(\overrightarrow{A}) = \max_{\tau \in J} \{C_\tau(\overrightarrow{A})\} \quad (3.7)$$
- *le flowtime (moyen) de $\mathcal{J}$ pour $\overrightarrow{A}$,*
$$C_{mean}(\overrightarrow{A}) = \frac{1}{\ell} C(\overrightarrow{A}) \text{ avec } C(\overrightarrow{A}) = \Sigma_{J \in \mathcal{J}} C_J(\overrightarrow{A}) \quad (3.8)$$
- *le makespan de $\mathcal{J}$ pour $\overrightarrow{A}$,*
$$C_{\max}(\overrightarrow{A}) = \max_{\nu_i \in \mathcal{N}} \{w_i(\overrightarrow{A})\} \quad (3.9)$$
- *le taux de disponibilité locale de $\overrightarrow{A}$,*
$$L(\overrightarrow{A}) = \Sigma_{\tau \in \mathcal{T}} \frac{\Sigma_{\rho \in \mathcal{R}_\tau,\ \nu(\tau, \overrightarrow{A}) \in l(\rho)} |\rho|}{\Sigma_{\rho \in \mathcal{R}_\tau} |\rho|} \quad (3.10)$$

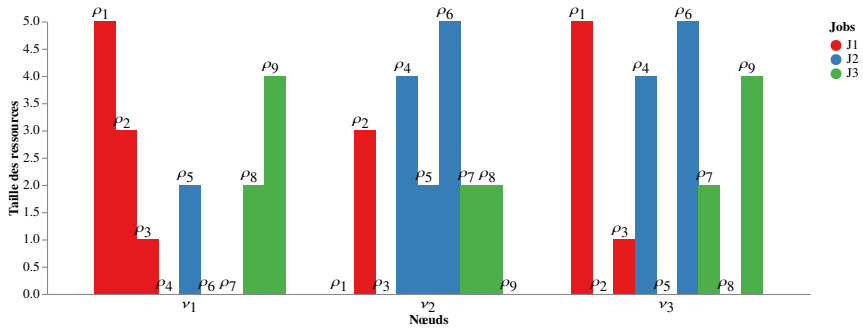
Contrairement au *makespan*, le *flowtime* dépend de l'ordre d'exécution des tâches sur chacun des nœuds. Le taux de disponibilité locale d'une allocation mesure la proportion des ressources traitées localement (équation (3.10)).

Exemple 3.7 (MASTA+). — À partir du système distribué $\mathcal{D} = \langle \mathcal{N}, \mathcal{E}, \mathcal{R} \rangle$ avec $\mathcal{N} = \{\nu_1, \nu_2, \nu_3\}$, $\mathcal{E} = \{(\nu_1, \nu_2), (\nu_1, \nu_3), (\nu_2, \nu_3)\}$ et $\mathcal{R} = \{\rho_1, \ldots, \rho_9\}$ où les ressources sont répliquées sur 2 nœuds (*cf.* figure 3.1a), nous considérons MASTA+ =

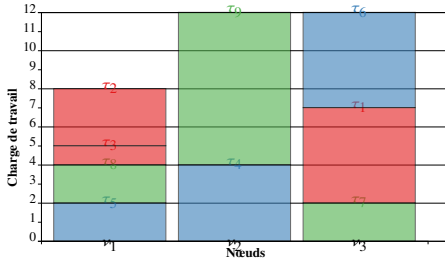
$\langle \mathscr{D}, \mathscr{T}, \mathscr{J}, c \rangle$ avec $\mathscr{T} = \{\tau_1, \ldots, \tau_9\}$ où chaque tâche τ_i nécessite la ressource ρ_i, $\mathscr{J} = \{J_1, J_2, J_3\}$ tel que $J_1 = \{\tau_1, \tau_2, \tau_3\}$, $J_2 = \{\tau_4, \tau_5, \tau_6\}$ et $J_3 = \{\tau_7, \tau_8, \tau_9\}$ et la fonction de coût donnée dans la table 3.1. Nous supposons que le coût d'une tâche est proportionnel à la taille des ressources et qu'il est deux fois plus important si la ressource est distante. Nous considérons ici l'allocation $\overrightarrow{A}$ (*cf.* figure 3.1b) avec $\overrightarrow{B}_1 = (\tau_5, \tau_8, \tau_3, \tau_2)$, $\overrightarrow{B}_2 = (\tau_4, \tau_9)$ et $\overrightarrow{B}_3 = (\tau_7, \tau_1, \tau_6)$. Le *makespan* et le *flowtime* sont $C_{max}(\overrightarrow{A}) = 12$ et $C(\overrightarrow{A}) = 8 + 12 + 12 = 32$.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9
$c(\tau, \nu_1)$	5	3	1	8	2	10	4	2	4
$c(\tau, \nu_2)$	10	3	2	4	2	5	2	2	8
$c(\tau, \nu_3)$	5	6	1	4	4	5	2	4	4

TABLE 3.1 – Le coût des tâches pour chacun des nœuds



(a) Répartition des ressources



(b) Allocation des tâches aux nœuds

FIGURE 3.1 – Distributions des ressources et des tâches pour notre exemple fil rouge

En résumé, le coût des tâches dépend du nœud qui l'exécute en raison de la localité des ressources. Notre objectif est de minimiser le *flowtime* moyen des jobs composés de tâches.

4. Consommation et réallocation

Nous décrivons ici les opérations de consommation et de réallocation ainsi que le protocole de négociation.

Une **consommation** de tâche consiste pour un nœud à supprimer une tâche de son lot pour l'exécuter. Cette opération modifie non seulement l'allocation courante mais également le problème MASTA+ sous-jacent où la tâche consommée n'est plus présente. La stratégie de consommation d'un agent spécifie l'ordonnancement des tâches pour le nœud dont il a la charge. Comme nous voulons minimiser le *flowtime* moyen des jobs, nous considérons ici une stratégie orientée job qui trie le lot d'abord par job puis par tâche au sein d'un même job (les tâches d'un même job sont consécutives dans le lot). En particulier, les jobs les moins coûteux sont prioritaires sur ceux plus coûteux pour minimiser localement le délai de réalisation des jobs. Par la suite, $\mathrm{J}_1 \blacktriangleleft_i \mathrm{J}_2$ signifie que les tâches de J_1 sont prioritaires sur celles de J_2 et $\tau_1 \lhd_i \tau_2$ que la tâche τ_1 est prioritaire sur la tâche τ_2. Plus formellement :

$$\begin{aligned} &\forall \tau_j, \tau_k \in \mathrm{B}_i \ \tau_j \prec_i \tau_k \Leftrightarrow \\ &\mathrm{job}(\tau_j) \blacktriangleleft_i \mathrm{job}(\tau_k) \vee (\mathrm{job}(\tau_j) = \mathrm{job}(\tau_k) \wedge \tau_j \lhd_i \tau_k) \end{aligned} \tag{4.1}$$

L'ajout ou la suppression d'une liste de tâches T dans le lot $\overrightarrow{\mathrm{B}}_i$ du nœud ν_i peuvent changer l'ordre d'exécution des tâches puisque ces opérations impliquent un réordonnancement du lot :

- $\overrightarrow{\mathrm{B}_i \oplus \mathrm{T}}$ désigne le lot qui contient l'ensemble des tâches $\mathrm{B}_i \cup \mathrm{T}$ trié selon $\prec_i$;
- $\overrightarrow{\mathrm{B}_i \ominus \mathrm{T}}$ désigne le lot qui contient $\mathrm{B}_i \setminus \mathrm{T}$ trié selon $\prec_i$.
- $\overrightarrow{\mathrm{B}_i \ominus \mathrm{T}_1 \oplus \mathrm{T}_2}$ désigne le lot qui contient $(\mathrm{B}_i \setminus \mathrm{T}_1) \cup \mathrm{T}_2$ trié selon $\prec_i$.

Une **réallocation bilatérale** est une opération qui modifie l'allocation courante via l'échange d'une ou de plusieurs tâches entre deux agents.

Définition 4.1 (Réallocation bilatérale). — *Soient MASTA+* $= \langle \mathscr{D}, \mathscr{T}, \mathscr{J}, c \rangle$ *un problème d'allocation et* $\overrightarrow{A} = (\overrightarrow{B}_1, \ldots, \overrightarrow{B}_m)$ *une allocation. La réallocation bilatérale de la liste non vide de tâches* T_1 *allouées au proposant* ν_i *en échange de la liste de tâches* T_2 *allouées au répondant* ν_j *dans* $\overrightarrow{A}$ *(*$\mathrm{T}_1 \subseteq B_i$ *et* $\mathrm{T}_2 \subseteq B_j$*) aboutit à l'allocation* $\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A})$ *avec les m lots* $\gamma_k(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A})$ *définis tels que :*

$$\gamma_k(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A}) = \begin{cases} \overrightarrow{B_i \ominus \mathrm{T}_1 \oplus \mathrm{T}_2} & \text{si } k = i, \\ \overrightarrow{B_j \ominus \mathrm{T}_2 \oplus \mathrm{T}_1} & \text{si } k = j, \\ \overrightarrow{B}_k & \text{sinon.} \end{cases} \tag{4.2}$$

Pour une réallocation bilatérale $\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{\mathrm{A}})$, on distingue deux cas :

- un **échange** où les deux listes de tâches sont non vides ($\mathrm{T}_1 \neq \emptyset \wedge \mathrm{T}_2 \neq \emptyset$), noté $\sigma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{\mathrm{A}})$;
- une **délégation** où un agent donne une partie de ses tâches à l'un de ses pairs sans contre-partie ($\mathrm{T}_2 = \emptyset$), notée $\delta(\mathrm{T}_1, \nu_i, \nu_j, \overrightarrow{\mathrm{A}})$. Si $|\mathrm{T}_1| = 1$, on parle de **délégation unaire**. Dans le cas contraire, on parle de **délégation *n*-aire**.

Nous verrons par la suite que la réallocation bilatérale de listes de tâches plutôt que d'ensembles permet de spécifier l'ordre selon lequel les tâches doivent être évaluées pour valider l'intérêt de tout ou partie de la réallocation.

Afin d'améliorer une allocation, nous introduisons la notion de réallocation bilatérale socialement rationnelle qui vérifie qu'une réallocation réduit le *flowtime* global, *i.e.* le délai de réalisation des jobs pour l'ensemble des nœuds.

Définition 4.2 (Réallocation bilatérale socialement rationnelle). — *Soient MASTA+* $= \langle \mathscr{D}, \mathscr{T}, \mathscr{J}, c \rangle$ *un problème d'allocation,* $\overrightarrow{A}$ *une allocation. La réallocation bilatérale* $\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A})$ *est socialement rationnelle vis-à-vis du flowtime global ssi le flowtime global décroît,*

$$C(\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A})) < C(\overrightarrow{A}) \tag{4.3}$$

Une allocation est dite **stable** s'il n'existe aucune réallocation bilatérale socialement rationnelle.

Contrairement à [3], nous ne considérons pas comme socialement rationnelles des réallocations qui réduisent le *flowtime* local (le délai de réalisation des jobs pour les seuls nœuds impliqués dans la réallocation) qui ne permet pas de garantir la convergence du processus de réallocation, ni même comme rationnelles des réallocations qui réduisent le *flowtime* local et le *makespan* (la charge maximale des agents). La réduction du *flowtime* global permet de garantir la terminaison du processus. Dorénavant lorsque nous parlerons de *flowtime* il s'agira, sauf précision, du *flowtime* global ($= \mathrm{C}(\overrightarrow{\mathrm{A}})$ défini équation (3.8)).

Propriété 4.3 (Terminaison). — *Soient MASTA+* $= \langle \mathscr{D}, \mathscr{T}, \mathscr{J}, c \rangle$ *un problème d'allocation et* $\overrightarrow{A}$ *une allocation qui n'est pas stable vis-à-vis du flowtime. Toute séquence de réallocations bilatérales socialement rationnelles issue de* $\overrightarrow{A}$ *est finie car elle atteint une allocation stable.*

Démonstration. — La preuve est similaire à celles du théorème 7 et du lemme 3 dans [11]. En effet, comme le nombre d'allocations est fini et $\Sigma_{\mathrm{J} \in \mathscr{J}} \mathrm{C}_\mathrm{J}(\overrightarrow{\mathrm{A}})$ décroît strictement à chaque étape, il ne peut y avoir qu'un nombre fini de réallocations socialement rationnelles aboutissant à une allocation stable. □

Pour réaliser des réallocations de tâches, les agents réalisent de multiples négociations bilatérales à un tour. Comme illustré sur la figure 4.1, chaque négociation, reposant sur un protocole d'offres alternées [21], inclut trois étapes de décision : (a) la

stratégie d'offre du proposant qui sélectionne une délégation, c'est-à-dire une liste de tâches dans son lot et un répondant (message `Propose`), (b) la stratégie de contre-offre qui permet au répondant de déterminer s'il décline la délégation (`Reject`), l'accepte (`Accept`), voire fait une contre-offre (`CounterPropose`) et (c) l'éventuelle réallocation est confirmée (`Confirm`) ou annulée (`Withdraw`) par le proposant selon les consommations qui ont eu lieu de manière concurrente.

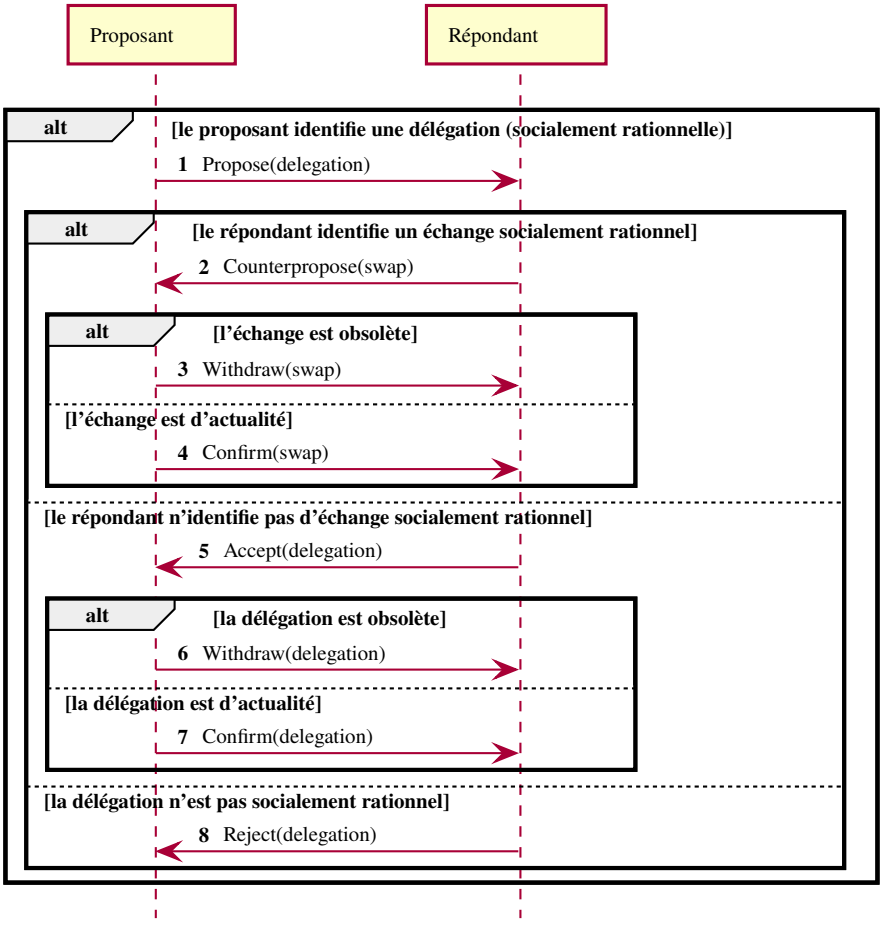
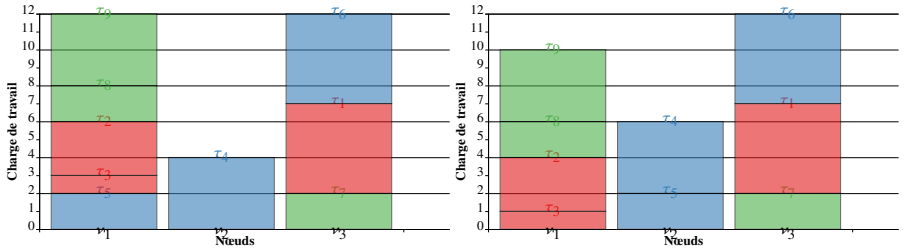


FIGURE 4.1 – Protocole de négociation bilatérale entre un agent « proposant » et un agent « répondant » pour la réallocation bilatérale de tâches.

Exemple 4.4 (Consommation et réallocation). — Considérons le problème MASTA+ de l'exemple 3.7, en particulier l'allocation $\overrightarrow{A}$ représentée dans la figure 3.1b. Selon la stratégie de consommation adoptée par les agents, chaque lot est trié par job, c'est-à-dire les jobs les moins coûteux sont prioritaires (*e.g.* $J_3 \blacktriangleleft_3 J_1 \blacktriangleleft_3 J_2$). L'ordre naturel sur les identifiants permet de défaire les égalités (*e.g.* $J_2 \blacktriangleleft_1 J_3$). Les tâches au



(a) Allocation après la délégation de τ_9 de ν_2 à ν_1

(b) Allocation après l'échange de τ_9 contre τ_5 entre ν_2 et ν_1

FIGURE 4.2 – Allocations de l'exemple fil rouge résultant de réallocations bilatérales

sein d'un même job sont triées par ordre de coût croissant ($\tau_3 \lhd_1 \tau_2$). La délégation de la tâche τ_9 par le nœud ν_2 au nœud ν_1 aboutissant à l'allocation $\overrightarrow{A'} = \delta([\tau_9], \nu_2, \nu_1, \overrightarrow{A})$ (*cf.* figure 4.2a) est socialement rationnelle vis-à-vis du *flowtime* car elle fait décroître le *flowtime* de 32 à 31 (les nouveaux temps de réalisation des jobs étant respectivement $C_{J_1}(\overrightarrow{A'}) = 7$, $C_{J_2}(\overrightarrow{A'}) = C_{J_3}(\overrightarrow{A'}) = 12$). Toutefois, l'échange de $\tau_9 \in B_2$ contre $\tau_5 \in B_1$ entre les nœuds ν_2 et ν_1, qui aboutit à l'allocation $\overrightarrow{A''} = \sigma([\tau_9], [\tau_5], \nu_2, \nu_1, \overrightarrow{A})$ (*cf.* figure 4.2b), est meilleur car il fait décroître le *flowtime* de 32 à 29 ($C_{J_1}(\overrightarrow{A''}) = 7$, $C_{J_2}(\overrightarrow{A''}) = 12, C_{J_3}(\overrightarrow{A''}) = 10$).

5. Stratégie de négociation

Nous décrivons ici les différentes parties de la stratégie de négociation et nous esquissons le comportement de l'agent lors du processus de négociation qui concerne les délégations de lots de tâches.

5.1. Modèle des pairs

Le modèle des pairs est construit à partir des informations échangées entre agents via des messages. En particulier, avant le processus de négociation et après chaque réallocation bilatérale dans laquelle il est impliqué, l'agent ν_i informe ses pairs de ce que chaque job J lui coûte ($c(J, \nu_i)$). Comme le nombre de jobs est négligeable par rapport au nombre de tâches, la taille de ces messages est dérisoire par rapport à la description des lots. Le modèle de la cible ν_k par le sujet ν_i repose sur :

(1) la base de croyances du sujet, éventuellement partielle ou obsolète, qui contient les croyances concernant les coûts des jobs pour ν_k ($c^i(J, \nu_k)$, $\forall J \in \mathcal{J}$) et donc les croyances concernant la charge de travail de ν_k ($w^i_k(\overrightarrow{A}) = \Sigma_{J \in \mathcal{J}} c^i(J, \nu_k)$) ;
(2) l'ordre des jobs dans le lot de la cible en se basant sur la stratégie de consommation commune, et sur la base de croyances.

Par souci de cohérence, on écrit $c^i(J, \nu_i) = c(J, \nu_i)$ et $w_i^i(\overrightarrow{A}) = w_i(\overrightarrow{A})$.

Le sujet peut alors déduire :

- la durée de réalisation ($C_J^i(\overrightarrow{B}_k)$) du job J pour une cible k, éventuellement lui-même ($\nu_k = \nu_i$), après l'ajout ($C_J^i(\overrightarrow{B_k \oplus T})$), la suppression ($C_J^i(\overrightarrow{B_k \ominus T})$) et le remplacement de tâches ($C_J^i(\overrightarrow{B_k \ominus T_1 \oplus T_2})$) ;
- la durée de réalisation d'un job J pour l'allocation,

$$C_J^i(\overrightarrow{A}) = \max_{\nu_k \in \mathcal{N}} C_J^i(\overrightarrow{B}_k) \text{ où } C_J^i(\overrightarrow{B}_i) = C_J(\overrightarrow{B}_i) \tag{5.1}$$

- le nœud **limitant** pour chaque job J, noté $\nu_{\max}^i(\overrightarrow{A}, J)$, *i.e.* le nœud ν_k pour lequel la durée de réalisation de ce job est la durée de réalisation maximale,

$$C_J^i(\overrightarrow{B}_k) = C_J^i(\overrightarrow{A}) \tag{5.2}$$

- le *flowtime* de l'allocation $\overrightarrow{A}$

$$C^i(\overrightarrow{A}) = \Sigma_{J \in \mathcal{J}} C_J^i(\overrightarrow{A}) \tag{5.3}$$

5.2. Règle d'acceptabilité

La **règle d'acceptabilité** est une décision locale prise par un agent impliqué dans une réallocation bilatérale, qui est basée sur ses connaissances et le modèle de ses pairs, pour accepter ou décliner la réallocation.

Définition 5.1 (Acceptabilité). — *Soient MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *un problème d'allocation et* $\overrightarrow{A}$ *une allocation. La réallocation bilatérale* $\gamma(T_1, T_2, \nu_i, \nu_j, \overrightarrow{A})$ *est acceptable par l'agent* $\nu_k \in \mathcal{N}$ *vis-à-vis du flowtime ssi l'agent croit que le flowtime décroît,*

$$\Sigma_{J \in \mathcal{J}} \max_{\forall \nu_o \in \mathcal{N} \setminus \{\nu_i, \nu_j\}} (C_J^k(\overrightarrow{B_i \ominus T_1 \oplus T_2}), C_J^k(\overrightarrow{B_j \ominus T_2 \oplus T_1}), C_J^k(B_o)) < C^k(\overrightarrow{A}) \tag{5.4}$$

L'acceptabilité vis-à-vis du *flowtime* repose sur la croyance à propos des durées de réalisation des jobs pour les nœuds avant et après la réallocation (équation (5.4)).

Nous proposons ici un processus où les agents initient des négociations bilatérales concurrentes qui doivent aboutir à des réallocations socialement rationnelles.

5.3. Stratégie d'offre

La **stratégie d'offre** d'un agent, qui repose sur ses connaissances, ses croyances et son modèle des pairs, identifie une délégation en trois étapes. Il s'agit donc pour un agent ν_i de sélectionner un lot d'offre, c'est-à-dire une liste de tâches à déléguer à un receveur dans un ensemble $\mathcal{N}'$ afin de réduire la durée de réalisation d'un job pour lequel il est limitant dans un ensemble $\mathcal{J}'$. Initialement, $\mathcal{J}' = \mathcal{J}$, $\mathcal{N}' = \mathcal{N}$.

(1) **Sélection d'un job.** Afin de réduire non seulement la durée de réalisation globale d'un job qui lui est affecté mais également celles des jobs qui suivent dans son lot, notre heuristique sélectionne le job J_* le plus prioritaire parmi ceux dont il est l'agent limitant,

$$\mathrm{J}_* = \min_{\blacktriangleleft_i}\{\mathrm{J} \in \mathrm{jobs}(\mathrm{B}_i) \cap \mathscr{J}' \mid \nu^i_{\max}(\overrightarrow{\mathrm{A}}, \mathrm{J} = \nu_i)\} \tag{5.5}$$

(2) **Sélection d'un receveur.** Les jobs d'un receveur impactés par la délégation sont ceux placés après J_* selon $\blacktriangleleft^i_j$. Afin de ne pas augmenter la durée de réalisation de ces jobs, notre heuristique sélectionne un receveur ν_* pour qui la somme des différences entre la durée de réalisation pour l'allocation et celle pour l'agent est la plus grande,

$$\nu_* = \mathit{random}\left\{\operatorname*{argmax}_{\nu_j \in \mathscr{N}'} \Sigma_{\mathrm{J}_* \blacktriangleleft^i_j \mathrm{J}}(\mathrm{C}^i_{\mathrm{J}}(\overrightarrow{\mathrm{A}}) - \mathrm{C}^i_{\mathrm{J}}(\overrightarrow{\mathrm{B}}_j))\right\} \tag{5.6}$$

où *random* est une fonction de choix aléatoire qui à chaque ensemble de nœuds associe un nœud de cet ensemble.

(3) **Sélection du lot d'offre.** Afin de déterminer le lot d'offre, on distingue une stratégie qui propose de déléguer une unique tâche comme dans [3] et une stratégie qui peut proposer plusieurs tâches.

   (a) **Sélection d'une délégation unaire.** Afin de réduire les durées de réalisation, l'initiateur sélectionne une tâche distante, c'est-à-dire dont la délégation réduira son coût d'exécution. Notre heuristique sélectionne la tâche du job J_* ou des jobs qui le précèdent dans $\overrightarrow{\mathrm{B}}_i$ avec le meilleur gain en terme de coût. En cas d'égalité, c'est la tâche prioritaire du lot qui est choisie,

$$\forall \mathscr{T}' \subseteq \mathscr{T}, \tau_* = \min_{\lhd_i}\left\{\operatorname*{argmax}_{\tau \in \mathscr{T}' \cap \mathrm{B}_i \cap \{\mathrm{J} \mid \mathrm{J}=\mathrm{J}_* \vee (\mathrm{J} \blacktriangleleft_i \mathrm{J}_*)\}} \mathrm{c}(\tau, \nu_i) - \mathrm{c}(\tau, \nu_*)\right\}$$

   La délégation $\delta([\tau_*], \nu_i, \nu_*, \overrightarrow{\mathrm{A}})$ est déclenchée si elle est acceptable pour l'initiateur (*cf.* définition 5.1).

   (b) **Sélection d'une délégation *n*-aire.** L'initiateur construit itérativement un lot d'offre T_*. Ce lot est une pile de tâches qui sera évaluée par la stratégie d'acceptation (*cf.* Section 5.4) pour que le receveur puisse refuser toute ou partie de ces tâches en dépilant le lot d'offre. Comme illustré dans l'algorithme 1, notre heuristique considère les tâches du job J_* ou des jobs qui le précèdent dans $\overrightarrow{\mathrm{B}}_i$ (ligne 2). L'initiateur ν_i sélectionne en priorité les tâches distantes, *i.e.* celles dont la délégation réduit le plus le coût d'exécution (lignes 3 et 6). Selon cet algorithme à cliquet, le délai de réalisation décroît strictement au cours de la construction du lot d'offre (ligne 8). De plus, la construction du lot d'offre est stoppée dès qu'une tâche candidate ne permet pas d'améliorer le délai de réalisation. Si le lot d'offre T_* est non vide, alors la délégation $\delta(\mathrm{T}_*, \nu_i, \nu_*, \overrightarrow{\mathrm{A}})$, qui est acceptable pour l'initiateur, est déclenchée.

Algorithme 1 : Construction du lot d'offre par l'initiateur ν_i

Entrées : J_* le job sélectionné dans l'étape 1 ;
ν_* le receveur sélectionné dans l'étape 2 ;

$T_* \leftarrow$ `empty_stack` ;
$T = \{\tau \mid \text{job}(\tau) = J_* \vee (\text{job}(\tau) \blacktriangleleft_i J_*)\}$;
$T' \leftarrow (\ldots, \tau^k, \ldots, \tau^l, \ldots) \mid \tau^i \in T \wedge (k < l \Leftrightarrow c(\tau^k, \nu_i) - c(\tau^k, \nu_*) > c(\tau^l, \nu_i) - c(\tau^l, \nu_*))$ `/* la liste des tâches par gain de coût d'exécution décroissant */`
bestFlowtime $= C^i(\overrightarrow{A})$;
tant que $T' \neq \emptyset$ **faire**
  $\tau_* \leftarrow$ `head`(T') ;
  $T' \leftarrow$ `tail`(T') ;
  si $C^i(\delta(T_* \cup \{\tau_*\}, \nu_i, \nu_*, \overrightarrow{A})) <$ *bestFlowtime* **alors**
    T_*.`push`(τ_*) ;
    bestFlowtime $\leftarrow C^i(\delta(T_*, \nu_i, \nu_*, \overrightarrow{A}))$;
  fin
  sinon
    Retourner T_* ;
  fin
fin
Retourner T_* ;

Quelque soit la stratégie de sélection du lot d'offre (3.a ou 3.b, si aucune délégation n'est déclenchée, la stratégie d'offre retourne dans l'étape 2 pour choisir un autre receveur ($\mathcal{N}' \leftarrow \mathcal{N}' \setminus \{\nu_*\}$). À défaut, la stratégie d'offre retourne dans l'étape 1 pour choisir un autre job ($\mathcal{J}' \leftarrow \mathcal{J}' \setminus \{J_*\}$). En cas d'échec, aucune délégation n'est proposée et l'agent passe en état de pause jusqu'à ce que sa base de croyances soit mise à jour et qu'une nouvelle opportunité (*i.e.* une délégation) soit trouvée.

5.4. Stratégie d'acceptation

Algorithme 2 : Sélection par le receveur ν_* du sous-lot parmi le lot d'offre reçu

Entrées : T_* le lot d'offre reçu

$T_{acc} \leftarrow T_*$;
tant que $\delta(T_{acc}, \nu_i, \nu_*, \overrightarrow{A})$ *n'est pas acceptable par l'agent* ν_* **faire**
  T_{acc}.`pop` ;
fin
Retourner T_{acc} ;

Selon la **stratégie d'acceptation**, le receveur accepte une délégation qu'il considère acceptable. Dans le cas contraire, il dépile une à une les tâches du lot d'offre T_* (*cf.* algorithme 2) pour éventuellement en accepter une partie. Quand le sous-lot T_{acc} est vide, le receveur refuse l'offre.

5.5. Comportement d'agent

Dans notre approche, une réallocation bilatérale de tâches est le résultat de négociations entre agents qui adoptent tous le même **comportement**. Les agents exécutent leur comportement selon leurs connaissances et croyances. Le comportement des agents est spécifié dans [5] par un automate fini déterministe[(1)]. Afin d'éviter les interblocages, les propositions sont associées à des dates butoirs. La base de croyances de l'agent est mise à jour lors de la réception des messages. Aucune proposition n'est soumise tant que l'agent croit que l'allocation est stable. Les comportements des agents leur permettent d'exécuter des négociations concurrentes mais les offres sont traitées de manière séquentielle.

Exemple 5.2 (Stratégie de négociation). — Considérons le problème MASTA+ de l'exemple 3.7 et l'allocation initiale $\overrightarrow{A}$ (*cf.* Figure 5.1a) telle que $\overrightarrow{B}_1 = (\tau_5, \tau_1)$, $\overrightarrow{B}_2 = (\tau_3, \tau_2, \tau_7, \tau_8, \tau_9)$ et $\overrightarrow{B}_3 = (\tau_4, \tau_6)$. Le *flowtime* est $C(\overrightarrow{A}) = 7 + 9 + 17 = 33$. On considère ici que les agents ont des croyances à jour. La stratégie d'offre de l'agent ν_2 sélectionne une délégation de la manière suivante :

(1) il sélectionne le job le plus prioritaire pour lequel il est limitant (*cf.* équation (5.5)), $J_* = J_3$;
(2) il sélectionne le receveur qui est le moins limitant pour J_3 (*cf.* équation (5.6)). Comme ni ν_1 ni ν_3 ne possèdent de tâche de J_3, l'agent ν_2 choisit aléatoirement, $\nu_* = \nu_1$;
(3) l'algorithme 1 permet à l'agent ν_2 de sélectionner son lot d'offre :
   (a) les tâches candidates, c'est-à-dire les tâches de J_3 ou les précédentes dans son lot de tâches, sont triées par gain de coût décroissant, $T' = [\tau_9, \tau_3, \tau_2, \tau_8, \tau_7]$;
   (b) la délégation de la tâche τ_9 améliore la durée de réalisation (*cf.* figure 5.1b),

$$C^{\,2}(\delta([\tau_9], \nu_2, \nu_1, \overrightarrow{A})) = 11 + 9 + 9 = 29 < 33 \tag{5.7}$$

   La tâche τ_9 est ajoutée au lot d'offre, $T_* = [\tau_9]$,
   (c) la délégation des tâches τ_3 et τ_9 améliore la durée de réalisation (*cf.* figure 5.1c),

$$C^{\,2}(\delta([\tau_9, \tau_3], \nu_2, \nu_1, \overrightarrow{A})) = 12 + 9 + 7 = 28 < 29 \tag{5.8}$$

   La tâche τ_3 est ajoutée au lot d'offre, $T_* = [\tau_9, \tau_3]$,

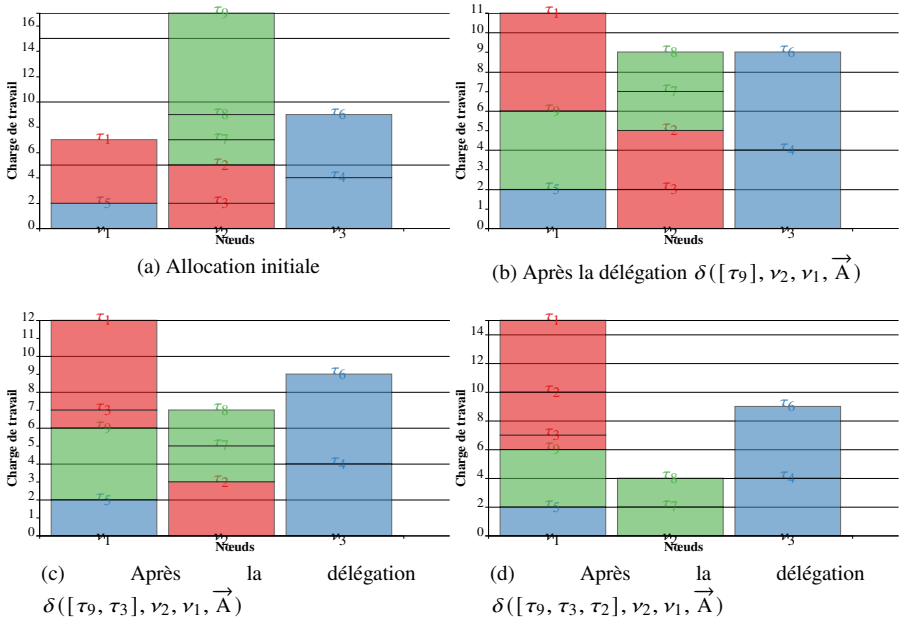
(1) https://gitlab.univ-lille.fr/maxime.morge/smastaplus/-/tree/master/doc/specification.

(d) la délégation des tâches τ_2, τ_3 et τ_9 n'améliore pas la durée de réalisation (*cf.* figure 5.1c),

$$C^2(\delta([\tau_9, \tau_3, \tau_2], \nu_2, \nu_1, \overrightarrow{A})) = 15 + 9 + 6 = 30 > 28 \quad (5.9)$$

Le lot d'offre sélectionné est $T_* = [\tau_9, \tau_3]$.

En résumé, l'agent ν_2 propose à ν_1 la délégation $\delta([\tau_9, \tau_3], \nu_2, \nu_1, \overrightarrow{A})$.



(a) Allocation initiale

(b) Après la délégation $\delta([\tau_9], \nu_2, \nu_1, \overrightarrow{A})$

(c) Après la délégation $\delta([\tau_9, \tau_3], \nu_2, \nu_1, \overrightarrow{A})$

(d) Après la délégation $\delta([\tau_9, \tau_3, \tau_2], \nu_2, \nu_1, \overrightarrow{A})$

FIGURE 5.1 – Allocations de l'exemple 5.2

6. ÉVALUATION EMPIRIQUE

Après une présentation des conditions expérimentales, nous comparons empiriquement notre approche avec une heuristique classique ainsi qu'avec les techniques d'optimisation distribuée sous contraintes (DCOP). Nous évaluons également notre nouveau critère de rationalité ainsi que les délégations n-aires[(2)].

6.1. CONDITIONS EXPÉRIMENTALES

L'application pratique que nous considérons est le déploiement distribué du patron de conception MapReduce pour le traitement de jeux de données massives sur une

[(2)]Ces expérimentations sont reproductibles à partir des instructions suivantes : https://gitlab.univ-lille.fr/maxime.morge/smastaplus/-/tree/master/doc/experiments.

grappe de serveurs, comme avec Spark [28]. Nous nous focalisons ici sur la phase *reduce* des jobs MapReduce que nous formalisons par un problème MASTA+ où plusieurs jobs sont soumis de façon concurrente. La fonction de coût est définie telle que :

$$c_i(\tau, \nu_j) = \Sigma_{\rho_j \in \mathscr{R}_\tau} c_i(\rho_j, \nu_j) \text{ avec } c_i(\rho_j, \nu_i) = \begin{cases} |\rho_j| \text{ si } \nu_i \in l(\rho_j) \\ \kappa \times |\rho_j| \text{ sinon} \end{cases} \tag{6.1}$$

où nous avons fixé empiriquement $\kappa = 2$ comme une valeur réaliste pour capturer le surcoût induit par la récupération des ressources distantes.

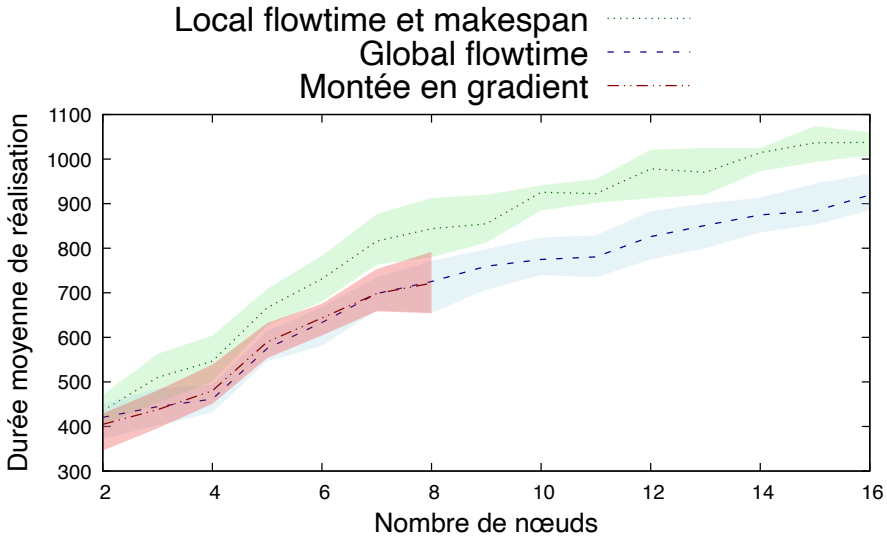
Notre prototype [5] est implémenté avec le langage de programmation Scala et la bibliothèque Akka [19] adaptée aux applications orientées messages, fortement concurrentes, distribuées et robustes. Nous supposons que : (a) le délai de transmission des messages est arbitraire mais non négligeable, (b) l'ordre des messages par paire émetteur-récepteur est préservé, (c) la distribution des messages est garantie. Les expériences ont été réalisées sur une lame munie de 20 CPUs avec 512 Go de RAM.

Ce travail est une première étape dans l'évaluation de nos stratégies, puisque nous comparons ici le calcul d'une réallocation, c'est-à-dire la résolution d'un problème MASTA+, sans les itérations induites par les consommations de tâches, même si la stratégie de consommation est nécessaire pour trier les lots de tâches de chaque agent. Nous considérons des instances de MASTA+ avec $m \in [2; 16]$ nœuds/agents, $\ell = 4$ jobs, $n = 3 \times \ell \times m$ tâches et une ressource par tâche. Chaque ressource ρ_i est répliquée 3 fois et $|\rho_i| \in [0; 100]$. Nous générons 10 instances de MASTA+, et pour chacune nous générons aléatoirement 10 allocations initiales. Nous évaluons les médianes et les déviations standards de trois métriques : (1) la durée moyenne de réalisation (équation (3.8)), (2) le taux de disponibilité locale (équation (3.10)), et (3) le temps de réordonnancement.

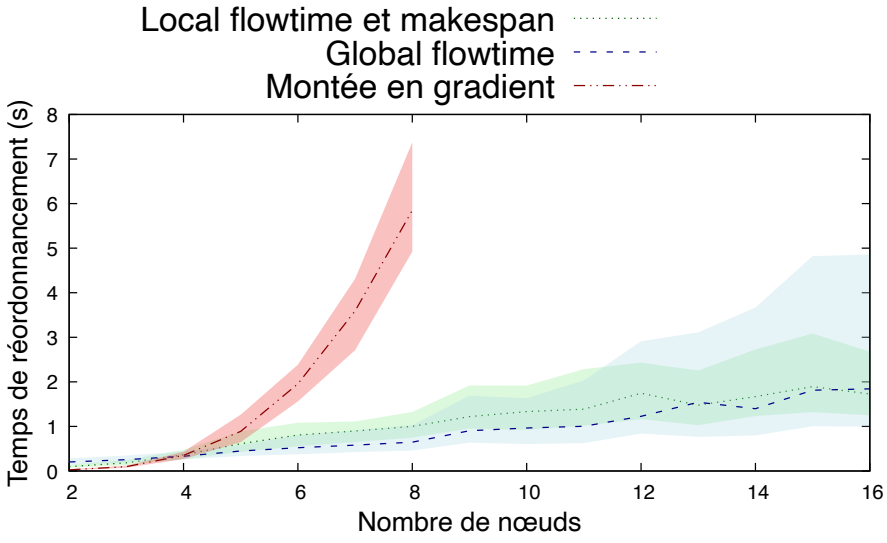
6.2. Heuristique classique et critère de rationalité

Les hypothèses que nous voulons tester sont : (1) la durée moyenne de réalisation atteinte par notre stratégie est proche de celle obtenue par l'approche classique ; (2) la décentralisation de notre approche permet d'accélérer le réordonnancement. De plus, contrairement à nos précédents travaux [3] qui utilisaient comme critère de rationalité le *flowtime* local et le *makespan*, nous considérons dans cet article uniquement le *flowtime* global qui suffit à garantir la convergence du processus de négociation. Nous souhaitons également vérifier que ce critère de rationalité permet d'améliorer significativement la qualité de la solution obtenue.

Les figures 6.1a et 6.1b comparent respectivement la durée moyenne de réallocation et le temps de réordonnancement de notre stratégie de délégation unaire avec celle présentée dans [3] ainsi qu'avec un algorithme de montée en gradient qui débutent avec la même allocation initiale générée aléatoirement. À chaque étape, l'algorithme de montée en gradient sélectionne parmi toutes les délégations possibles celle qui minimise la durée moyenne de réalisation.



(a) Durée moyenne de réalisation des jobs



(b) Temps de réordonnancement

FIGURE 6.1 – Durée moyenne de réalisation et temps de réordonnancement pour notre stratégie, celle proposée dans [3] et l'algorithme de montée en gradient

Dans la figure 6.1a, nous observons que si la qualité des solutions obtenues par la stratégie proposée dans [3] est légèrement inférieure à celle fournie par l'algorithme de montée en gradient, notre stratégie atteint desormais des solutions de qualité similaire. Cette différence s'explique par le fait qu'une réallocation socialement rationnelle vis-à-vis du *flowtime* global ne peut que strictement améliorer la durée moyenne de réalisation, ce qui n'est pas le cas lorsque l'on utilise le *flowtime* local comme critère de rationalité. De plus, le fait que la réduction du *makespan* ne soit plus imposée par le critère de rationalité permet d'élargir le nombre de délégations possibles pouvant améliorer la durée moyenne de réalisation.

Dans la figure 6.1b, nous observons que le temps de réordonnancement pour notre nouvelle stratégie reste approximativement identique et donc largement meilleur que celui de l'algorithme de montée en gradient qui croît exponentiellement avec le nombre de noeuds. Le critère de rationalité proposé dans cet article permet donc d'améliorer significativement la durée moyenne de réalisation avec un temps de réordonnancement similaire. Il est à noter que l'algorithme de montée en gradient n'a été évalué que sur des instances MASTA+ de petites tailles en raison de son temps de réordonnancement rédhibitoire. On peut s'attendre à avoir un temps de réordonnancement plus grand avec une méthode de recherche locale telle que le recuit simulé sans pour autant avoir de garanties sur la qualité du résultat. Par conséquent, même si le nombre d'agents est faible, le gain réalisé sur la durée moyenne de réalisation par réordonnancement via l'algorithme de montée en gradient sera pénalisé et annulé par le surcoût du temps de réordonnancement. Ce surcoût pénalise l'équilibrage en continu des charges dans un système distribué qui devrait s'adapter aux phénomènes perturbateurs (consommation de tâches, libération de jobs, ralentissement des nœuds).

La figure 6.2 compare le taux de disponibilité locale de l'allocation initiale et des allocations obtenues par notre stratégie, par celle proposée dans [3] et par l'algorithme de montée en gradient. On observe que le taux de disponibilité des allocations obtenues par nos stratégies est proche de celui obtenu avec l'algorithme de montée en gradient. Même si, contrairement à ce dernier, notre stratégie n'évalue pas toutes les délégations unaires possibles, elle s'avère efficace en sélectionnant les tâches distantes dont la délégation réduit le coût.

La figure 6.3 compare le temps de réordonnancement de notre stratégie avec un ou plusieurs *threads* d'exécution et celui de l'algorithme de montée en gradient. Comme la version de notre stratégie avec plusieurs *threads* est exécutée sur plusieurs cœurs, nous observons que l'accélération de notre algorithme augmente en fonction du nombre d'agents. Par exemple, avec une durée moyenne de réalisation similaire (si on néglige le non déterminisme observable de leurs exécutions), la version *multi-threads* est 10 fois plus rapide que la version *mono-thread* pour 16 agents.

6.3. Délégation n-aires

Nous souhaitons ici vérifier que les délégations *n*-aires permettent de réduire le temps de réordonnancement.

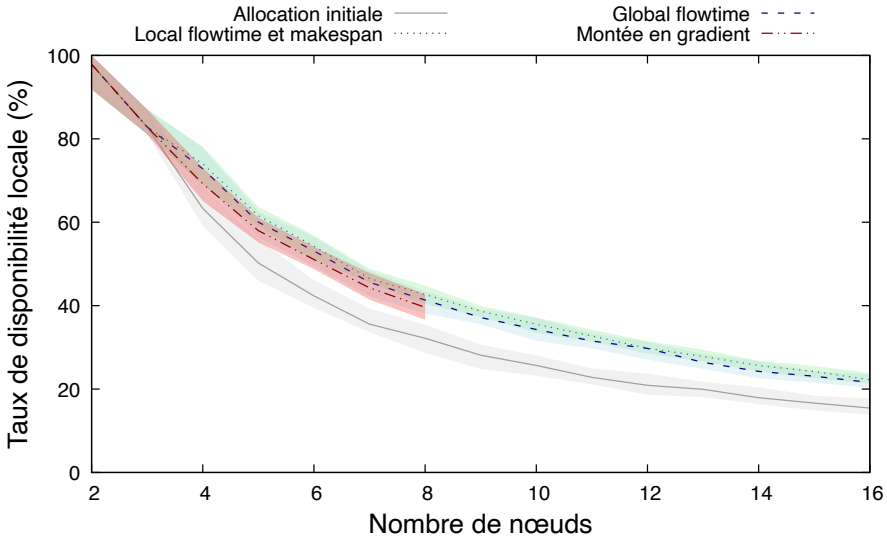


FIGURE 6.2 – Taux de disponibilité locale de l'allocation initiale et des allocations obtenues par notre stratégie, par celle proposée dans [3] et par l'algorithme de montée en gradient

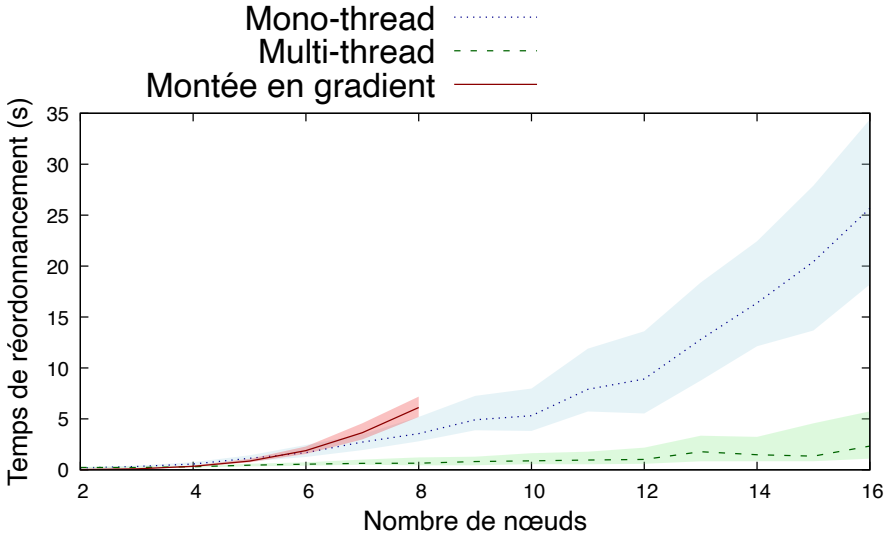


FIGURE 6.3 – Temps de réordonnancement de notre stratégie avec un ou plusieurs *threads* d'exécution et de l'algorithme de montée en gradient

Les figures 6.4a et 6.4b comparent respectivement la durée moyenne de réallocation et le temps de réordonnancement de notre stratégie de délégation unaire avec celle notre stratégie de délégation n-aire. Nous observons que, si la durée moyenne de réalisation atteinte par la stratégie de délégation n-aire est légèrement moins bonne que pour la stratégie de délégation unaire, le gain en terme de temps de réordonnancement est en revanche très avantageux.

La figure 6.5 représente l'évolution de la durée moyenne de réalisation de ces deux stratégies de sélection du lot d'offre pour une instance quelconque non équilibrée de problème de réallocation. On observe que la sélection d'une délégation n-aire réduit le nombre de délégations (ici 40 plutôt que 66) pour atteindre des allocations stables dont les durées moyennes de réalisation sont similaires. En conséquence, la sélection d'une délégation n-aire réduit le temps de réordonnancement (ici 0, 35 seconde plutôt que 1, 8 seconde).

6.4. Optimisation distribuée sous contraintes

Nous souhaitons ici vérifier que le temps de réordonnancement de notre stratégie de négociation est beaucoup plus faible que celui obtenu par des techniques d'optimisation distribuée sous contraintes (DCOP) et qu'elle permet d'atteindre des durées de réalisation plus faibles.

Trouver l'allocation optimale est un problème d'ordonnancement qui minimise le *flowtime* de ℓ jobs, constitués de n tâches, sur m nœuds indépendants. Il peut être modélisé mathématiquement de la manière suivante :

(1) n variables de décision x_i telles que,

$$x_i = (o-1) \times n + k \text{ si } \tau_i \text{ est la } k^{\text{e}} \text{ tâche à partir de la fin sur } \nu_o \tag{6.2}$$

(2) n^2 contraintes pour que chaque tâche soit affectée à une unique position

$$\forall i \in [1,n] \forall j \in [1,n] \setminus \{i\}\ x_i \neq x_j; \tag{6.3}$$

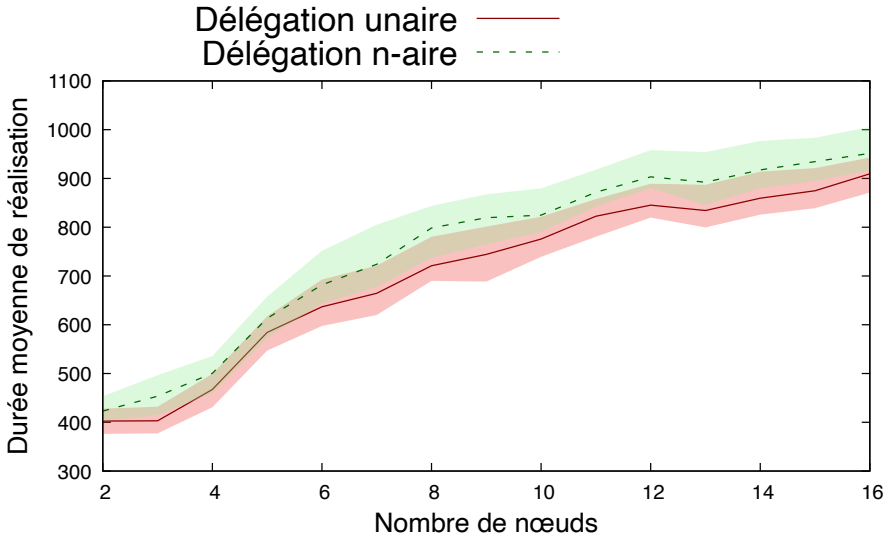
(3) la fonction objectif à minimiser est $\mathrm{C}(\overrightarrow{\mathrm{A}})$.

$$\begin{aligned} \mathrm{C}(\overrightarrow{\mathrm{A}}) &= \Sigma_{\mathrm{J} \in \mathscr{J}} \mathrm{C}_{\mathrm{J}}(\overrightarrow{\mathrm{A}}) \\ &= \Sigma_{\mathrm{J} \in \mathscr{J}} \max_{\nu_i \in \mathcal{N}} C_{last(\nu_i, \mathrm{J})}(\vec{A}) \end{aligned} \tag{6.4}$$

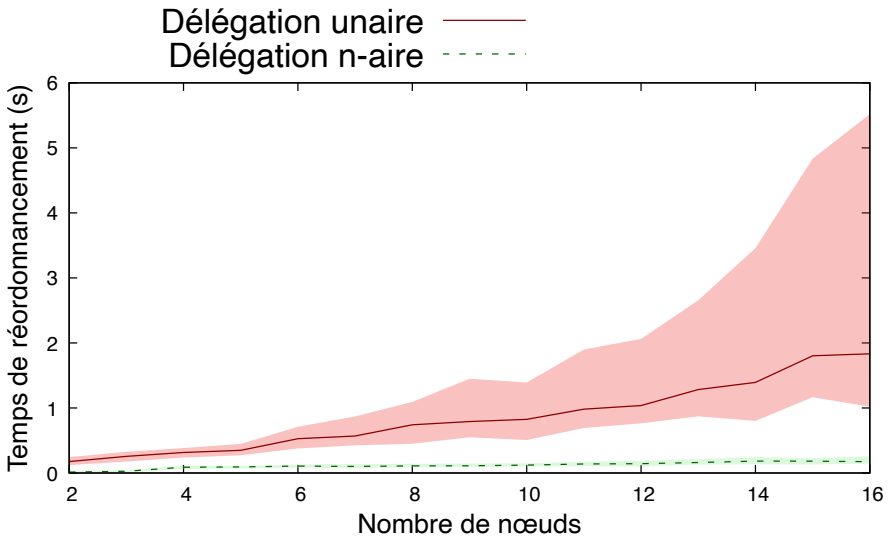
où $last(\nu_i, \mathrm{J})$ est la dernière tâche du job J dans le lot de tâches de $\overrightarrow{\mathrm{B}}_i$:

$$last(\nu_i, \mathrm{J}) = \underset{\tau_k \in \mathrm{J} \cap \mathrm{B}_i}{\operatorname{argmax}} \operatorname{card}(\{\tau \in \overrightarrow{\mathrm{B}}_i \mid \tau \prec_i \tau_k\}) \tag{6.5}$$

Pour résoudre ce problème, nous considérons l'algorithme MGM2 [20] – Maximum Gain Message – comme le plus adapté, car c'est un algorithme distribué de recherche locale qui est approché et asynchrone. Nous avons utilisé la bibliothèque pyDCOP [22, 23]. Nous considérons 100 problèmes de réallocation pour des instances de MASTA+ avec $m = 2$ nœuds, $\ell = 4$ jobs et $n = \ell \times m = 8$ tâches.

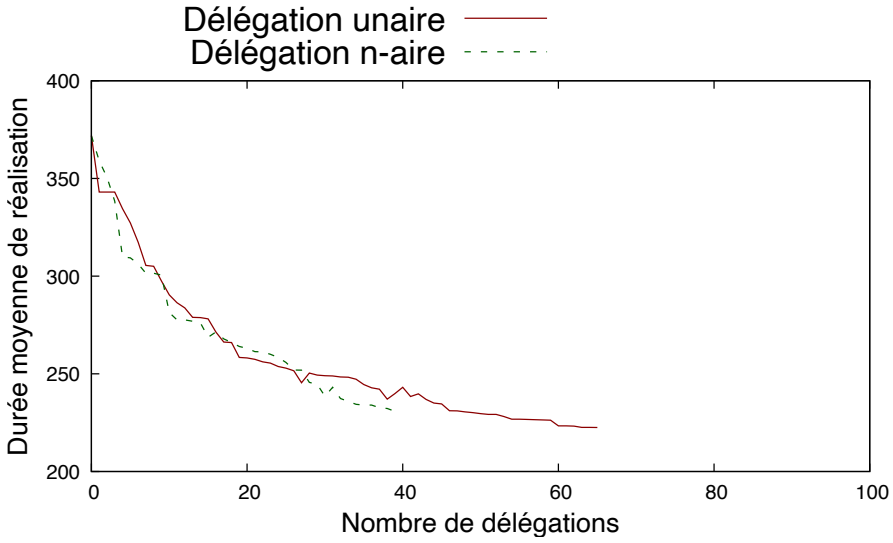


(a) Durée moyenne de réalisation

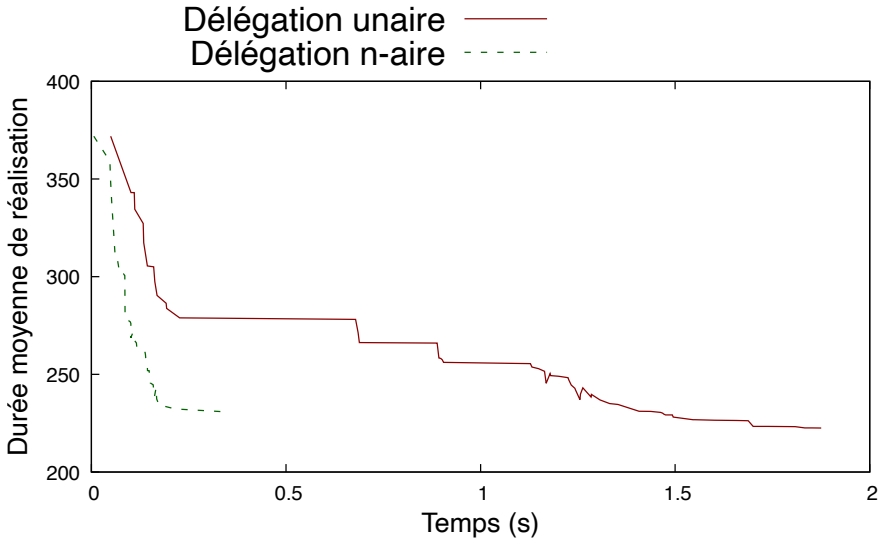


(b) Temps de réordonnancement

FIGURE 6.4 – Durée moyenne de réalisation et temps d'ordonnancement pour notre stratégie de délégation unaire et notre stratégie de délégation *n*-aire



(a) Durée moyenne de réalisation des stratégies de délégation unaire et n-aire au cours des délégations



(b) Durée moyenne de réalisation de la stratégie de délégation unaire et n-aire au cours du temps

FIGURE 6.5 – Durée moyenne de réalisation des stratégies de délégation unaire et n-aire au cours des délégations (en haut) et du temps (en bas)

La figure 6.6 compare les durées moyennes de réalisation pour des réallocations obtenues par notre stratégie en 55 millisecondes (en moyenne) avec l'algorithme MGM2 dont le *timeout* est respectivement 2, 5 et 10 secondes. On peut noter que l'algorithme MGM2 propose systématique une réallocation quand le *timeout* est de 5 ou 10 secondes mais jamais avec un *timeout* de 2 secondes. Dans ce cas, nous considérons que l'allocation initiale générée de manière pseudo-aléatoire est conservée. Au-delà des temps de réordonnancement qui peuvent s'expliquer par le fait que MGM2 est implémenté en Python et notre stratégie s'exécute sur la machine virtuelle Java [13], nos expériences montrent que même si le *timeout* est de 5 secondes, l'algorithme MGM2 retourne une réallocation où la durée moyenne de réalisation est supérieure à celle obtenue par notre stratégie. Accroître le *timeout* ne permet pas d'améliorer la durée moyenne de réalisation des allocations retournées par MGM2. Notons que l'algorithme MGM2 ne propose jamais de réallocation avec $m = 3$ nœuds, $\ell = 5$ jobs et $n = 3 \times \ell \times m = 45$ tâches quand le *timeout* est de 15 minutes. Cet algorithme ne passe pas l'échelle sur ce type de problème.

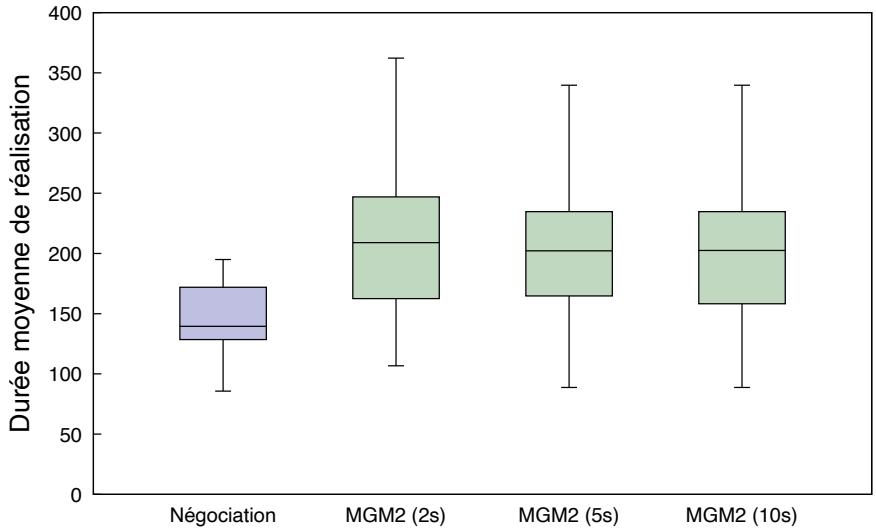


FIGURE 6.6 – Diagrammes en boîte des durées moyennes de réalisation pour les réallocations obtenues par notre stratégie en 55 millisecondes (en moyenne) avec l'algorithme MGM2 dont le *timeout* est respectivement 2, 5 et 10 secondes

7. CONCLUSION

Dans cet article, nous avons proposé un système multi-agents pour la réallocation de tâches sur des nœuds en fonction de la localisation des ressources nécessaires afin de réduire la durée moyenne de réalisation de jobs concurrents. Nos expériences montrent que la durée moyenne de réalisation atteinte par notre stratégie est proche de

celle atteinte par l'approche heuristique classique et qu'elle réduit considérablement le temps de réordonnancement. Cela est dû au fait que le processus de négociation adapte en continu l'allocation afin d'améliorer l'équilibre des charges en réduisant les durées de réalisation des jobs pour les nœuds qui sont limitants. D'une part, la stratégie de consommation consiste à exécuter d'abord les tâches des jobs les moins coûteux avant celles des jobs les plus coûteux. D'autre part, la stratégie de délégation consiste à sélectionner un job qui permet de réduire les durées de réalisation du proposant en choisissant un répondant qui n'est pas limitant pour les jobs impactés et en choisissant des tâches distantes dont la délégation réduit le coût d'exécution. Nous avons également comparé notre approche avec les techniques d'optimisation distribuée sous contraintes (DCOP), en particulier l'algorithme MGM2 pour lequel les temps de réordonnancement et les durées moyennes de réalisation sont très supérieures.

Une analyse de sensibilité pour étudier l'influence du facteur de réplication va au-delà de la portée de cet article, mais mériterait une étude approfondie. Notre approche passe l'échelle, car elle gère un grand nombre de tâches grâce à des décisions locales des agents à propos de la prochaine tâche à déléguer/exécuter. En outre, le surcoût de la négociation est négligeable par rapport au bénéfice de l'équilibrage des charges, car aucune négociation n'est déclenchée lorsque les agents estiment que l'allocation est stable.

Comme notre cadre de négociation le permet, nous envisageons d'ajouter une stratégie de contre-proposition pour sélectionner une contre-partie susceptible de déclencher des échanges de tâches afin d'améliorer la durée moyenne de réalisation des jobs. D'une manière générale, les travaux futurs doivent étendre le processus de réallocation des tâches vers un processus itératif, dynamique et continu, qui se déroule de manière concurrente à l'exécution des tâches, pour permettre au système distribué de s'adapter aux phénomènes perturbateurs (consommation de tâches, libération de jobs, ralentissement des nœuds).

Remerciements

Nous adressons nos remerciements aux relecteurs pour leur travail minutieux et leurs précieux conseils.

Bibliographie

[1] Q. Baert, A.-C. Caron, M. Morge, J.-C. Routier & K. Stathis, « An adaptive multi-agent system for task reallocation in a MapReduce job », *Journal of Parallel and Distributed Computing* **153** (2021), p. 75-88.

[2] E. Beauprez, L. Bigand, A.-C. Caron, M. Morge & J.-C. Routier, « Réaffectation de tâches de la théorie à la pratique : état de l'art et retour d'expérience », in *Vingt-neuvièmes journées francophones sur les systèmes multi-agents, JFSMA 2021, Bordeaux, France, 28-30 juin 2021*, JFSMA, Cépaduès, 2021, p. 51-60.

[3] E. Beauprez, A.-C. Caron, M. Morge & J.-C. Routier, « Une stratégie de négociation multi-agents pour réduire la durée moyenne de réalisation », in *Vingt-neuvièmes journées francophones sur les systèmes multi-agents, JFSMA 2021, Bordeaux, France, 28-30 juin 2021*, JFSMA, Cépaduès, 2021, p. 31-40.

[4] ———, « A Multi-Agent Negotiation Strategy for Reducing the Flowtime », in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence – Volume 1: ICAART,*, INSTICC, SciTePress, 2021, p. 58-68.

[5] E. Beauprez & M. Morge, « Scala implementation of the Extended Multi-agents Situated Task Allocation », `https://gitlab.univ-lille.fr/maxime.morge/smastaplus`, 2020.

[6] A. Beynier, F. Charpillet, D. Szer & A.-I. Mouaddib, « DEC-MDP / DEC-POMDP », in *Markov Decision Processes in Artificial Intelligence* (O. S. Olivier Buffet, éd.), Wiley-ISTE, 2010, p. 277-313.

[7] J. Bruno, E. G. Coffman Jr. & R. Sethi, « Scheduling Independent Tasks to Reduce Mean Finishing Time », *Commun. ACM* **17** (1974), nº 7, p. 382-387.

[8] B. Chen, C. N. Potts & G. J. Woeginger, « A review of machine scheduling: Complexity, algorithms and approximability », in *Handbook of combinatorial optimization* (D.-Z. Du & P. M. Pardalos, éds.), Springer, 1998, p. 1493-1641.

[9] H.-L. Choi, L. Brunet & J. P. How, « Consensus-based decentralized auctions for robust task allocation », *IEEE transactions on robotics* **25** (2009), nº 4, p. 912-926.

[10] R. W. Conway, W. L. Maxwell & L. W. Miller, *Theory of Scheduling*, Addison- Wesley, Reading, MA, 1967.

[11] U. Endriss, N. Maudet, F. Sadri & F. Toni, « Negotiating Socially Optimal Allocations of Resources », *Journal of Artificial Intelligence Research* **25** (2006), p. 315-348.

[12] F. Fioretto, E. Pontelli & W. Yeoh, « Distributed constraint optimization problems and applications: A survey », *Journal of Artificial Intelligence Research* **61** (2018), p. 623-698.

[13] B. Fulgham & I. Gouy, « The Computer Language Benchmarks Game », `https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html`, 2021.

[14] W. Horn, « Minimizing average flow time with parallel machines », *Operations Research* **21** (1973), nº 3, p. 846-847.

[15] Y. Jiang, « A survey of task allocation and load balancing in distributed systems », *IEEE Transactions on Parallel and Distributed Systems* **27** (2016), nº 2, p. 585-599.

[16] H. W. Kuhn, « The Hungarian method for the assignment problem », *Naval research logistics quarterly* **2** (1955), nº 1-2, p. 83-97.

[17] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. N. Prasad, A. Raja, R. Vincent & X. Q. Xuan, P. Zhang, « Evolution of the GPGP/TAEMS domain-independent coordination framework », *Autonomous agents and multi-agent systems* **9** (2004), nº 1-2, p. 87-143.

[18] S. Li, R. R. Negenborn & G. Lodewijks, « A Distributed Constraint Optimization Approach for Vessel Rotation Planning », in *Computational Logistics*, Springer, 2014, p. 61-80.

[19] Lightbend, « Akka is the implementation of the Actor Model on the JVM », `http://akka.io`, 2020.

[20] R. T. Maheswaran, J. P. Pearce & M. Tambe, « Distributed Algorithms for DCOP: A Graphical-Game-Based Approach », in *Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems, September 15-17, 2004, The Canterbury Hotel, San Francisco, California, USA* (D. A. Bader & A. A. Khokhar, éds.), ISCA, 2004, p. 432-439.

[21] A. Rubinstein, « Perfect Equilibrium in a Bargaining Model », *Econometrica* **50** (1982), nº 1, p. 97-109.

[22] P. Rust & G. Picard, « pyDCOP is a python library for Distributed Constraints Optimization », `https://github.com/Orange-OpenSource/pyDcop`, 2021.

[23] P. Rust, G. Picard & F. Ramparany, « Mise en place d'une décision collective résiliente sur une infrastructure IoT à l'aide du framework PyDCOP (démonstration) », in *Vingt-sixièmes journées francophones sur les systèmes multi-agents, JFSMA 2018, Métabief, France, 10-12 Octobre 2018*, JFSMA, Cépaduès, 2018, p. 223-224.

[24] A. Schaerf, Y. Shoham & M. Tennenholtz, « Adaptive Load Balancing : A Study in Multi-Agent Learning », *Journal of Artificial Intelligence Research* **2** (1995), p. 475-500.

[25] O. Shehory & S. Kraus, « Methods for task allocation via agent coalition formation », *Artificial Intelligence* **101** (1998), nº 1-2, p. 165-200.

[26] J. Turner, Q. Meng, G. Schaefer & A. Soltoggio, « Distributed Strategy Adaptation with a Prediction Function in Multi-Agent Task Allocation », in *Proc. of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2018, p. 739-747.

[27] M. P. Wellman, « A market-oriented programming environment and its application to distributed multicommodity flow problems », *Journal of Artificial Intelligence Research* **1** (1993), p. 1-23.

[28] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker & I. Stoica, « Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing », in *Proc. of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI) ; San Jose, CA, USA*, 2012, p. 15-28.

Abstract. — In this paper, we study the problem of task reallocation for load-balancing in distributed data processing models that tackle vast amount of data. We propose a strategy based on cooperative agents used to optimize the rescheduling of tasks in multiple jobs which must be executed as soon as possible. It allows agents to determine locally the next tasks to process, to delegate, eventually to swap according to their knowledge, their own belief base and their peer modelling. The novelty lies in the ability of agents to identify opportunities and bottleneck agents, and afterwards to reassign some bundles of tasks thanks to concurrent bilateral negotiations. The strategy adopted by the agents allows to warrant a continuous improvement of the flowtime. Our experimentation reveals that our strategy reaches a flowtime which is better than the one reached by a DCOP resolution, close to the one reached by the classical heuristic approach, and significantly reduces the rescheduling time.

Keywords. — Multi-Agents Systems, Distributed Problem Solving, Agent-based Negotiation.

Manuscrit reçu le 21 mars 2022, accepté le 23 novembre 2022.