Une stratégie de négociation multi-agents pour réduire la durée moyenne de réalisation

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

Une stratégie de négociation multi-agents pour réduire la durée moyenne de réalisation

Ellie Beauprez
Ellie.Beauprez@univ-lille.fr

Anne-Cécile Caron
Anne-Cecile.caron@univ-lille.fr

Maxime Morge
Maxime.Morge@univ-lille.fr

Jean-Christophe Routier
Jean-Christophe.Routier@univ-lille.fr

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

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 de multiples jobs devant être exécutés le plus tôt possible. Elle permet à un agent de déterminer localement la prochaine tâche à exécuter ou à déléguer grâce à ses connaissances, ses croyances et son modèle des pairs. La nouveauté réside dans la capacité des agents à identifier les opportunités et les agents limitants pour réallouer efficacement les tâches à travers des négociations bilatérales concurrentes. Nos expérimentations montrent que la durée moyenne de réalisation atteinte par notre stratégie reste proche de celle obtenue avec une heuristique classique mais que le temps d'ordonnancement est significativement réduit.

Mots-clés : *Résolution collective de problèmes, Négociation multi-agents*

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 for multiple jobs which must be executed as soon as possible. It allows an agent to determine locally the next task to process or to delegate according to its knowledge, its own belief base and its peer modelling. The novelty lies in the ability of agents to identify opportunities and bottleneck agents, and afterwards to reassign some of the tasks thanks to concurrent bilateral negotiations. Our experimentation reveals that our strategy reaches a flowtime which is close to the one reached by the classical heuristic approach and significantly reduces the rescheduling time.

Keywords: *Distributed Problem Solving, Agent-based Negotiation*

1 Introduction

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.-à-d. des ensembles de tâches) concurrents doivent êtres exécutés le plus tôt possible, et (b) les ressources (c.-à-d. 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.-à-d. le patron de conception Map-Reduce [19]. 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 [4] et de l'équilibrage de charge dans les systèmes distribués [9]. La plupart d'entre eux adoptent l'approche orientée marché [18, 1, 17] et modélisent le problème comme un jeu non-coopératif. Par contraste, nous supposons comme [2] que les agents sont coopératifs, c.-à-d. ils ont une perception locale et partielle de l'allocation, mais ils partagent le même objectif plutôt que maximiser leur satisfaction individuelle. Nous allons ici au-delà en considérant plusieurs jobs composés d'ensembles de tâches, chacune pouvant être exécutée par un seul des agents, tous compétents. Les agents souhaitent minimiser la durée moyenne de réalisation des jobs, c.-à-d. le *flowtime* moyen. La principale difficulté provient du fait que la réaffectation d'une tâche ne modifie pas uniquement la charge de travail des agents impliqués mais

également leur propre ordonnancement.

Nous proposons une stratégie de délégation des tâches, qui décide quelle transaction est suggérée ou acceptée. Elle repose sur un modèle des pairs et détermine 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 une tâche et un receveur. La règle d'acceptabilité détermine si l'agent accepte ou refuse une délégation.

Nos contributions sont les suivantes [1].
(a) Nous formalisons le problème d'allocation multi-agents de jobs composés de tâches situées dont les coûts diffèrent selon la localisation des ressources.
(b) 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 des tâches.
(c) Nous avons conduit des expériences approfondies qui montrent que notre méthode atteint un *flowtime* proche de celui obtenu par l'heuristique classique et réduit significativement le temps de réordonnancement.

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 quelle tâche négocier et avec qui. Notre évaluation empirique est décrite dans la section 6. La section 7 résume notre contribution et présente nos perspectives.

2 Travaux connexes

Les problèmes d'ordonnancement classiques sont des problèmes d'optimisation sous contraintes qui peuvent être approximés par différentes heuristiques telles que la méthode de montée en gradient ou le recuit simulé (voir [13] pour une synthèse). Ces approches ne sont pas toujours adaptées à la réallocation de tâches dans des systèmes distribués où décentralisation et adaptativité sont nécessaires. En effet, d'une part, 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.

Pour ces raisons l'ordonnancement multi-agents a reçu une attention particulière [4], notamment pour les problèmes d'équilibrage de charge dans des systèmes distribués [9]. Banerjee et Hecker proposent dans [3] un protocole général distribué d'allocation de ressources pour l'équilibrage de charge de jobs à gros grains sur un système massivement distribué. Ils soulignent que les interactions locales entre agents font émerger des propriétés globales. En revanche, Selvitopi et al. se penchent dans [16] sur l'ordonnancement simultané de tâches de granularité plus fine, dans le but d'améliorer la localité des données et d'équilibrer les charges de travail. Leur approche est basée sur des modèles de graphes et d'hypergraphes qui utilisent des connaissances spécifiques à l'application. Notre étude vise à lier les deux niveaux de granularité par la réallocation de tâches indépendantes au sein de jobs multiples sans disposer d'information a priori sur ces jobs et tâches.

Schaerf et al. se penchent dans [15] sur le comportement adaptatif d'agents pour un équilibrage de charge efficace utilisant l'apprentissage multi-agent par renforcement. Turner et al. combinent dans [17] l'apprentissage par classification supervisée avec un processus interne de prise de décision pour l'assignation de tâche. À l'inverse, nous ne considérons aucun modèle préalable, ni des données, ni de l'environnement, car cela ne serait pas pertinent par rapport à la classe d'applications pratiques qui nous concerne.

Dans [10], les auteurs distinguent deux types de proximité : la localité des ressources qui sont requises pour l'exécution des tâches et le voisinage des agents. Sur ce dernier point, nous supposons un réseau social complètement connecté avec des coûts de communications uniformes.

D'une part, la plupart des travaux adoptent l'approche orientée marché et modélisent le problème comme un jeu non-coopératif. Par exemple, An et al. proposent dans [1] un mécanisme distribué de négociation où des agents égoïstes négocient sur des ressources à la fois

1. Cet article est une synthèse (60 %) en français de [6].

le prix contractuel et la pénalité de dégagement. D'autre part, la plupart des algorithmes d'optimisation sous contraintes distribuées (voir [12] pour une synthèse récente) tentent d'optimiser une fonction mono-objectif utilitaire (une somme des coûts à minimiser). Plus récemment, [2] vise un objectif égalitaire qui est la minimisation du temps nécessaire à la réalisation de l'ensemble des jobs (c.-à-d. le *makespan*). 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.

3 Tâches situées

Dans cette section, nous étendons le modèle formel de [2] pour considérer plusieurs jobs concurrents composés d'ensembles 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 1 (Système distribué)**.** *Un système est un triplet $\mathcal{D} = \langle \mathcal{N}, \mathcal{E}, \mathcal{R} \rangle$ où :*

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

$$l : \mathcal{R} \to 2^{\mathcal{N}} \tag{1}$$

Pour plus de simplicité, nous faisons l'hypothèse ici qu'il y a exactement un agent par nœud, que toutes les ressources sont accessibles pour tous les agents et que l'exécution des tâches est fiable.

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

Définition 2 (Job/Tâche)**.** *Soient $\mathcal{D}$ un système distribué et Res l'espace des résultats. On considère un ensemble de ℓ jobs $\mathcal{J} = \{J_1, \ldots, J_\ell\}$. Chaque job J_i associé à la date de libération $t^0_{J_i}$ est un ensemble de k_i tâches $J_i = \{\tau_1, \ldots, \tau_{k_i}\}$ où chaque tâche τ est une fonction $\tau : 2^{\mathcal{R}} \to Res$.*

On note $\mathcal{T} = \bigcup_{1 \leq i \leq \ell} \mathrm{J}_i$ l'ensemble des n tâches sous-jacentes à $\mathcal{J}$ et $\mathcal{R}_\tau \subseteq \mathcal{R}$ 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, $|\mathcal{J}| << |\mathcal{T}|$.

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

Définition 3 (Coût)**.** *Soient $\mathcal{D}$ un système distribué et $\mathcal{T}$ un ensemble de tâches. La fonction de coût $c : \mathcal{T} \times \mathcal{N} \to \mathbb{R}^*_+$ est telle que :*

$$c(\tau, \nu_i) \leq c(\tau, \nu_j) \Leftrightarrow \quad \Sigma_{\rho \in \mathcal{R}_\tau, \nu_i \in l(\rho)} |\rho| > \Sigma_{\rho \in \mathcal{R}_\tau, \nu_j \in l(\rho)} |\rho| \tag{2}$$

Comme la collecte de ressources distantes représente un surcoût, une tâche est moins coûteuse si les ressources nécessaires sont « plus locales » (cf. section 6).

Nous considérons le problème d'allocation multi-agents de jobs composés de tâches situées.

Définition 4 (MASTA+)**.** *Un problème d'allocation multi-agents de jobs est un quadruplet $MASTA+ = \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ où :*

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

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

Définition 5 (Allocation)**.** *Une allocation pour un problème MASTA+ est un vecteur de m lots de tâches ordonnées $\vec{A} = ((B_1, \prec_1), \ldots, (B_m, \prec_m))$ où chaque lot $(B_i, \prec_i)$ est l'ensemble des tâches ($B_i \subseteq \mathcal{T}$) affectées au nœud ν_i associé à un ordre total strict ($\prec_i \subseteq \mathcal{T} \times \mathcal{T}$). $\tau_j \prec_i \tau_k$ signifie que si $\tau_j, \tau_k \in B_i$ alors τ_j est exécutée avant τ_k par ν_i. L'allocation $\vec{A}$ vérifie :*

$$\forall \tau \in \mathcal{T},\ \exists \nu_i \in \mathcal{N},\ \tau \in B_i \tag{3}$$

$$\forall \nu_i \in \mathcal{N}, \forall \nu_j \in \mathcal{N} \setminus \{\nu_i\},\ B_i \cap B_j = \emptyset \tag{4}$$

Toutes les tâches sont allouées (Eq. 3) et chacune n'est allouée qu'à un seul nœud (Eq. 4). Par souci de concision, on note :

- $\vec{\mathrm{B}}_i = (\mathrm{B}_i, \prec_i)$, le lot trié de ν_i ;

— $\min_{\prec_i} \mathrm{B}_i$, la prochaine tâche à exécuter par ν_i :
— $\mathrm{jobs}(\mathrm{B}_i)$, l'ensemble des jobs affectés à ν_i, i.e. les jobs ayant au moins une tâche dans B_i ;
— $\nu(\tau, \vec{\mathrm{A}})$, le nœud chargé de τ dans $\vec{\mathrm{A}}$;
— $\mathrm{w}_i(\vec{\mathrm{A}}) = \Sigma_{\tau \in \mathrm{B}_i} \mathrm{c}(\tau, \nu_i)$, la charge de travail du nœud ν_i pour l'allocation $\vec{\mathrm{A}}$.

Comme on suppose que les nœuds 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 :

$$\mathrm{C}_\tau(\vec{\mathrm{A}}) = \mathrm{t}(\tau, \nu(\tau, \vec{\mathrm{A}})) + \mathrm{c}(\tau, \nu(\tau, \vec{\mathrm{A}})) \quad (5)$$
$$\text{avec } \mathrm{t}(\tau, \nu_i) = \Sigma_{\tau' \in \mathrm{B}_i | \tau' \prec_i \tau} \mathrm{c}(\tau', \nu_i)$$

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 et leur date d'achèvement, et le *makespan* qui est le temps nécessaire à la réalisation de l'ensemble des jobs.

Définition 6 (*Flowtime/Makespan*)**.** *Soient MASTA+ un problème d'allocation de tâches et $\vec{A}$ une allocation. On définit :*

— *la durée de réalisation de $J \in \mathcal{J}$ pour $\vec{A}$,*

$$C_J(\vec{A}) = \max_{\tau \in J} \{C_\tau(\vec{A})\} \quad (6)$$

— *le flowtime moyen de $\mathcal{J}$ pour $\vec{A}$,*

$$C(\vec{A}) = \frac{1}{\ell} \Sigma_{J \in \mathcal{J}} C_J(\vec{A}) \quad (7)$$

— *le makespan de $\mathcal{J}$ pour $\vec{A}$,*

$$\max_{\nu_i \in \mathcal{N}} \{w_i(\vec{A})\} \quad (8)$$

— *le taux de disponibilité locale de $\vec{A}$,*

$$L(\vec{A}) = \sum_{\tau \in \mathcal{T}} \frac{\sum_{\rho \in \mathcal{R}_\tau,\, \nu(\tau, \vec{A}) \in l(\rho)} |\rho|}{\sum_{\rho \in \mathcal{R}_\tau} |\rho|} \quad (9)$$

Contrairement au *makespan*, le *flowtime* dépend de l'ordre d'exécution des tâches sur chacun des nœuds. Le taux de disponibilité locale mesure la proportion des ressources traitées localement (Eq. 9).

Exemple 1 (MASTA+)**.** *À partir de $\mathcal{D} = \langle \mathcal{N}, \mathcal{E}, \mathcal{R} \rangle$ 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. Fig. 1a), nous considérons MASTA+ $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ avec $\mathcal{T} = \{\tau_1, \ldots, \tau_9\}$ où chaque tâche τ_i nécessite la ressource ρ_i, $\mathcal{J} = \{J_1, J_2, J_3\}$ 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 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. L'allocation représentée sur la figure 1b est telle que $\vec{B}_1 = (\tau_1, \tau_4, \tau_7)$, $\vec{B}_2 = (\tau_5, \tau_8, \tau_2)$ et $\vec{B}_3 = (\tau_3, \tau_9, \tau_6)$. Le makespan et le flowtime moyen sont $C_{max}(\vec{A}) = 10$ et $C(\vec{A}) = 8.33$.*

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.

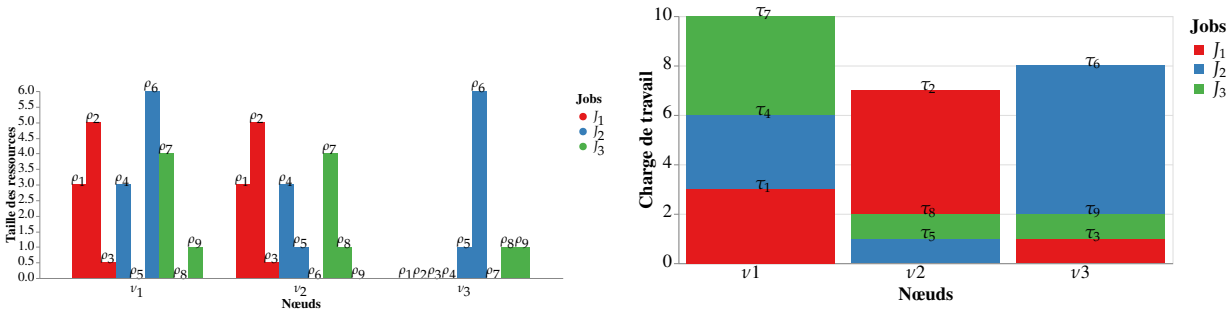
4 Consommation et délégation

Nous décrivons ici les opérations de consommation/délégation et le protocole de négociation.

L'ajout ou la suppression d'une tâche τ dans le lot $\vec{\mathrm{B}}_i$, qui modifient l'ensemble des tâches, peuvent changer l'ordre d'exécution des tâches.
Si $\tau \notin \mathrm{B}_i$ alors $\overrightarrow{\mathrm{B}_i \oplus \tau}$ désigne le lot qui contient l'ensemble des tâches $\mathrm{B}_i \cup \{\tau\}$ trié selon $\prec_i$.
Si $\tau \in \mathrm{B}_i$ alors $\overrightarrow{\mathrm{B}_i \ominus \tau}$ désigne le lot qui contient $\mathrm{B}_i \setminus \{\tau\}$ trié selon $\prec_i$.
Ces opérations impliquent un réordonnancement du lot.

La stratégie de consommation d'un agent spécifie l'ordonnancement des tâches dont il a la charge. Son étude va au-delà de la portée de cet article [5]. 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 tâches des jobs les moins coûteux sont prioritaires sur celles des jobs les 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.

La délégation d'une tâche est un évènement disruptif qui modifie l'allocation courante ; c'est



(a) Répartition des ressources

(b) Allocation des tâches aux nœuds

FIGURE 1 – Distribution des ressources et des tâches pour notre exemple fil rouge

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9
$c(\tau,\nu_1)$	3	5	0.5	3	2	6	4	2	1
$c(\tau,\nu_2)$	3	5	0.5	3	1	12	4	1	2
$c(\tau,\nu_3)$	6	10	1	6	1	6	8	1	1

TABLE 1 – Le coût des tâches pour chacun des nœuds

une réallocation.

Définition 7 (Délégation)**.** *Soient MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *un problème d'allocation et* $\vec{A} = (\vec{B}_1, \ldots, \vec{B}_m)$ *une allocation. Si la tâche* τ *est allouée au donneur* ν_i *(*$\tau \in B_i$*), alors la délégation au receveur* ν_j *aboutit à l'allocation* $\delta(\tau, \nu_i, \nu_j, \vec{A})$ *avec les m lots* $\delta(\tau, \nu_i, \nu_j, \vec{B}_k)$ *définis tels que :*

$$\delta(\tau, \nu_i, \nu_j, \vec{B}_k) = \begin{cases} \overrightarrow{B_i \ominus \tau} & si\ k = i \\ \overrightarrow{B_j \oplus \tau} & si\ k = j \\ \vec{B}_k & sinon \end{cases} \quad (10)$$

Une délégation de tâche doit réduire le *makespan* local et le *flowtime* local.

Définition 8 (Délégation socialement rationnelle)**.** *Soient MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *un problème d'allocation,* $\vec{A}$ *une allocation et* $\delta(\tau, \nu_i, \nu_j, \vec{A})$ *l'allocation après la délégation de* τ *par* ν_i *à* ν_j*. Cette délégation est socialement rationnelle :*
— vis-à-vis du makespan ssi le makespan local décroît,

$$w_j(\vec{A}) + c(\tau, \nu_j) < w_i(\vec{A}) \quad (11)$$

— vis-à-vis du flowtime ssi le flowtime local décroît,

$$\sum_{J \in jobs(B_i \cup B_j)} \max(C_J(\overrightarrow{B_i \ominus \tau}), C_J(\overrightarrow{B_j \oplus \tau})) < \sum_{J \in jobs(B_i \cup B_j)} \max(C_J(\vec{B}_i), C_J(\vec{B}_j)) \quad (12)$$

Une allocation est stable s'il n'existe aucune délégation socialement rationnelle.

Dans une allocation stable vis-à-vis du *makespan*, les agents ne peuvent plus améliorer localement le *makespan*.

Propriété 1 (Terminaison)**.** *Soient MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *un problème d'allocation et* $\vec{A}$ *une allocation qui n'est pas stable vis-à-vis du makespan. Il existe un chemin fini de délégations socialement rationnelles vis-à-vis de ce critère qui mène à une allocation stable pour le makespan.*

Cette propriété découle du théorème 7 dans [8]. En revanche, une séquence de délégations socialement rationnelles vis-à-vis du *flowtime* ne mène pas nécessairement à une allocation stable pour ce critère.

Pour réaliser des délégations de tâches, les agents réalisent de multiples négociations bilatérales à un tour. Chaque négociation, reposant sur un protocole d'offres alternées [14], inclut trois étapes de décision : (a) l'offre de stratégie du

proposant qui sélectionne une délégation potentielle, c.-à-d. une tâche dans son lot et un receveur, (b) une règle d'acceptabilité qui permet au répondant de déterminer s'il décline ou accepte la délégation, et (c) dans ce dernier cas, la délégation est confirmée ou annulée par le proposant selon les consommations qui ont eu lieu entre-temps.

5 Stratégie de délégation

Nous décrivons ici les différentes parties de la stratégie de délégation et nous esquissons le comportement de l'agent.

Au delà de la connaissance des agents sur le problème (i.e. la localisation des ressources et donc le coût des tâches pour les nœuds), 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 délégation dans laquelle il est impliqué, l'agent ν_i informe ses pairs que chaque job J lui coûte :

$$\mathrm{c}(\mathrm{J},\nu_i) = \Sigma_{\tau \in \mathrm{J} \cap \mathrm{B}_i} \mathrm{c}(\tau,\nu_i) \ \forall \mathrm{J} \in \mathcal{J} \qquad (13)$$

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_j par le sujet ν_i repose sur :
(a) la base de croyances du sujet, éventuellement partielle ou obsolète, qui contient les croyances concernant les coûts des jobs pour ν_j ($\mathrm{c}^i(\mathrm{J},\nu_j)$, $\forall \mathrm{J} \in \mathcal{J}$) et les croyances concernant la charge de travail de ν_j ($\mathrm{w}^i_j(\vec{\mathrm{A}}) = \Sigma_{\mathrm{J} \in \mathcal{J}} \mathrm{c}^i(\mathrm{J},\nu_j)$) ;
(b) la stratégie de consommation de la cible supposée par le sujet, notée $(\mathcal{J}, \blacktriangleleft^i_j)$.
Par souci de lisiblité, on écrit $\mathrm{c}^i(\mathrm{J},\nu_i) = \mathrm{c}(\mathrm{J},\nu_i)$ et $\mathrm{w}^i_i(\vec{\mathrm{A}}) = \mathrm{w}_i(\vec{\mathrm{A}})$.

Le sujet peut alors déduire la durée de réalisation du job J pour la cible, éventuellement après l'ajout/la suppression de τ ($\mathrm{C_J}^i(\vec{\mathrm{B}}_j)$, $\mathrm{C_J}^i(\overrightarrow{\mathrm{B}_j \oplus \tau})$ et $\mathrm{C_J}^i(\overrightarrow{\mathrm{B}_j \ominus \tau})$) ainsi que la durée de réalisation d'un job pour l'allocation, $\mathrm{C_J}^i(\vec{\mathrm{A}}) = \max_{\nu_j \in \mathcal{N}} \mathrm{C_J}^i(\vec{\mathrm{B}}_j)$ où $\mathrm{C_J}^i(\vec{\mathrm{B}}_i) = \mathrm{C_J}(\vec{\mathrm{B}}_i)$.

Le sujet considère que la cible est agent limitant pour le job J, noté $\nu_j = \nu^i_{\max}(\vec{\mathrm{A}},\mathrm{J})$, si la durée de réalisation de ce job pour la cible est la durée de réalisation maximale, c.-à-d. $\mathrm{C_J}^i(\vec{\mathrm{A}}) = \mathrm{C_J}^i(\vec{\mathrm{B}}_j)$.

La **règle d'acceptabilité** est une décision locale prise par le receveur d'une délégation, qui est basée sur ses connaissances et le modèle de ses pairs, pour accepter ou décliner une délégation.

Définition 9 (Acceptabilité)**.** *La délégation $\delta(\tau,\nu_i,\nu_j,\vec{A})$ de la tâche τ du donneur ν_i au receveur ν_j dans $\vec{A}$ est acceptable par le receveur :*
— vis-à-vis du makespan ssi le receveur croit que le makespan local décroît,

$$w_j(\vec{A}) + c(\tau,\nu_j) < w^j_i(\vec{A}) \qquad (14)$$

— vis-à-vis du flowtime ssi le receveur croit que le flowtime local décroît,

$$\Sigma_{J \in \mathcal{J}} \max(C^j_J(\overrightarrow{B_i \ominus \tau}), C_J(\overrightarrow{B_j \oplus \tau})) < \Sigma_{J \in \mathcal{J}} \max(C^j_J(\vec{B}_i), C_J(\vec{B}_j)) \qquad (15)$$

Alors que le premier critère s'appuie sur la croyance concernant la charge de travail du donneur (Eq. 14), le second repose sur la connaissance des durées de réalisation des jobs pour le receveur avant et après la délégation ainsi que ses croyances concernant les durées de réalisation pour le donneur avant et après (Eq. 15).

Pour réduire le *flowtime* moyen, la règle d'acceptabilité vérifie non seulement l'acceptabilité vis-à-vis du *flowtime* mais aussi l'acceptabilité vis-à-vis du *makespan* afin de garantir la convergence des négociations (Prop. 1).

La **stratégie d'offre** d'un donneur ν_i, qui repose sur ses connaissances, ses croyances et son modèle des pairs, sélectionne une délégation en quatre étapes.
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,

$$\forall \mathcal{J}' \subseteq \mathcal{J}, \qquad (16)$$
$$\mathrm{J}_* = \min_{\blacktriangleleft_i} \{\mathrm{J} \in \mathrm{jobs}(\mathrm{B}_i) \cap \mathcal{J}' \mid \nu_i = \nu^i_{\max}(\vec{\mathrm{A}},\mathrm{J})\}$$

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 le 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,

$$\forall \mathcal{N}' \subseteq \mathcal{N}, \quad (17)$$
$$\nu_* = \min_{<}\{\operatorname{argmax}_{\nu_j \in \mathcal{N}'} \sum_{\mathrm{J}_* \blacktriangleleft_j^i \mathrm{J}} (\mathrm{C_J}^i(\vec{\mathrm{A}}) - \mathrm{C_J}^i(\vec{\mathrm{B}}_j))\}$$

où $<$ désigne l'ordre naturel sur les identifiants des nœuds.

3. Sélection d'une tâche. Afin de réduire les durées de réalisation, le donneur sélectionne une tâche distante dont la délégation réduira son coût puisqu'elle sera exécutée localement. Notre heuristique sélectionne donc la tâche du job J_* ou des jobs qui le précèdent dans $\vec{\mathrm{B}}_i$ avec le meilleur gain. En cas d'égalité, c'est la tâche prioritaire du lot qui est choisie,

$$\forall \mathcal{T}' \subseteq \mathcal{T}, \tau_* = \quad (18)$$
$$\min_{\lhd_i}\{\operatorname{argmax}_{\tau \in \mathcal{T}' \cap \mathrm{B}_i \cap \{\mathrm{J} | \mathrm{J}=\mathrm{J}_* \vee (\mathrm{J} \blacktriangleleft_i \mathrm{J}_*)\}} \mathrm{c}(\tau, \nu_i) - \mathrm{c}(\tau, \nu_*)\}$$

4. Validation. Par symétrie avec le critère d'acceptabilité (cf. Def. 9), le critère de déclenchement est une décision locale prise par le donneur pour déterminer si la délégation est perçue comme étant socialement rationnelle. Dans le but de garantir la convergence du processus de négociation, la règle de déclenchement est une conjonction du critère de déclenchement vis-à-vis du *makespan* et de celui vis-à-vis du *flowtime*. Si la règle de déclenchement n'est pas vérifiée, une autre tâche ($\mathcal{T}' = \mathcal{T}' \setminus \{\tau_*\}$ dans l'étape 3) ou un autre receveur ($\mathcal{N}' = \mathcal{N}' \setminus \{\nu_*\}$ dans l'étape 2), éventuellement un autre job ($\mathcal{J}' = \mathcal{J}' \setminus \{\mathrm{J}_*\}$ dans l'étape 1) est choisi. 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.

Dans notre approche, une réallocation de tâche est le résultat de négociations entre agents qui adoptent tous le même **comportement** : ils alternent entre les rôles de proposant, de répondant et de contractant. Les agents exécutent leur comportement selon leurs connaissances et croyances. Le comportement des agents est spécifié dans [7] par un automate fini déterministe [2]. Afin d'éviter les interblocages, les propositions sont associées à des dates butoirs.

La réception de message venant des pairs met à jour la base de croyances de l'agent et aucune proposition n'est envoyée tant que l'agent croit que l'allocation est stable.

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

Exemple 2 (Stratégie de délégation)**.** *Considérons le problème MASTA+ et l'allocation $\vec{A}$ de l'exemple 1. Nous supposons que les agents ont des croyances à jour et qu'ils savent qu'ils adoptent tous la même stratégie de consommation. Contrairement à ν_1 et ν_2, l'agent ν_3 peut faire une proposition en sélectionnant :*
1. *$J_* = J_2$ est le job pour lequel il est agent limitant (Eq. 16) comme illustré sur la figure 1b ;*
2. *$\nu_* = \nu_2$ est l'agent le moins limitant pour les jobs impactés (Eq. 17) que sont le job J_3 pour ν_1 et les jobs J_1 et J_3 pour ν_2 :*

$$\sum_{J_* \blacktriangleleft_1^3 J} C_J^3(\vec{A}) - C_J^3(\vec{B}_1) = 0 < \sum_{J_* \blacktriangleleft_2^3 J} C_J^3(\vec{A}) - C_J^3(\vec{B}_2) = 8 \quad (19)$$

3. *$\tau_* = \tau_3$ est la tâche avec le gain le plus important (Eq. 18) dans J_2 ou dans les jobs antérieurs, i.e. J_1 et J_3 :*

$$\begin{aligned} c(\tau_6, \nu_3) - c(\tau_6, \nu_2) &= 6 - 12 = -6 \\ c(\tau_9, \nu_3) - c(\tau_9, \nu_2) &= 1 - 2 = -1 \\ c(\tau_3, \nu_3) - c(\tau_3, \nu_2) &= 1 - 0.5 = 0.5 \end{aligned} \quad (20)$$

4. *la délégation est déclenchable puisque le critère de déclenchement vis-à-vis du flowtime est vérifié,*

$$\begin{aligned} &\Sigma_{J \in \mathcal{J}} \max(C_J(\overrightarrow{B_3 \ominus \tau_3}), C_J^3(\overrightarrow{B_2 \oplus \tau_3})) = 16.5 \\ &< \Sigma_{J \in \mathcal{J}} \max(C_J(B_3), C_J^3(B_2)) = 17.0 \end{aligned} \quad (21)$$

et le critère de déclenchement vis-à-vis du makespan est vérifié,

$$w_2^3(\vec{A}) + c(\tau, \nu_2) = 7.5 < w_3(\vec{A}) = 8.0 \quad (22)$$

Le donneur ν_3 délègue la tâche τ_3 à l'agent ν_2 pour atteindre l'allocation $\vec{A}' = \delta(\tau_3, \nu_3, \nu_2, \vec{A})$ telle que $\vec{B}_1' = \{\tau_1, \tau_4, \tau_7\}$, $\vec{B}_2' = \{\tau_5, \tau_8, \tau_3, \tau_2\}$ et $\vec{B}_3' = \{\tau_9, \tau_6\}$. L'allocation obtenue est stable car il n'existe aucune délégation socialement rationnelle.

6 Évaluation empirique

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 grappe de serveurs, comme avec Spark [19]. Nous nous focalisons ici sur la phase *reduce* des jobs MapReduce. Cela peut être formalisé par un problème MASTA+ où plusieurs jobs sont soumis de façon concurrente et la fonction de coût est telle que :

$$\begin{aligned} c_i(\tau, \nu_j) &= \Sigma_{\rho_j \in \mathcal{R}_\tau} c_i(\rho_j, \nu_j) \\ \text{avec } c_i(\rho_j, \nu_i) &= \begin{cases} |\rho_j| \text{ si } \nu_i \in l(\rho_j) \\ \kappa \times |\rho_j| \text{ sinon} \end{cases} \end{aligned} \quad (23)$$

où nous avons fixé empiriquement $\kappa = 2$ comme une valeur réaliste.

Notre prototype [7] est implémenté avec le langage de programmation Scala et la bibliothèque Akka [11] adaptée aux applications orientées messages, fortement concurrentes, distribuées et robustes. Comme le suggère notre application pratique, nous supposons que : (a) le délai de transmission des messages est arbitraire mais non négligeable, (b) l'ordre des messages par pair émetteur-récepteur est préservé, (c) la distribution des messages est garantie, (d) le graphe d'accointance est connexe. Les expériences ont été réalisées sur une lame munie de 20 CPUs avec 512Go de RAM.

Ce travail est une première étape dans l'évaluation de nos stratégies, puisque nous comparons ici le calcul d'une réallocation, c.-à-d. 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 trois métriques : (1) le *flowtime* moyen (Eq. 7), (2) le taux de disponibilité locale (Eq. 9), et (3) le temps d'ordonnancement. Nous cherchons à (a) comparer les durées moyennes de réalisation des jobs atteintes par notre SMA avec celles atteintes par les approches classiques, et (b) évaluer l'accélération grâce à la décentralisation.

Les résultats atteints par le solveur polyvalent IBM® ILOG® CPLEX® lorqu'il résout le problème sous-jacent d'optimisation mathématique discrète non linéaire sont médiocres. C'est la raison pour laquelle nous comparons notre stratégie de négociation 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 le *flowtime* moyen.

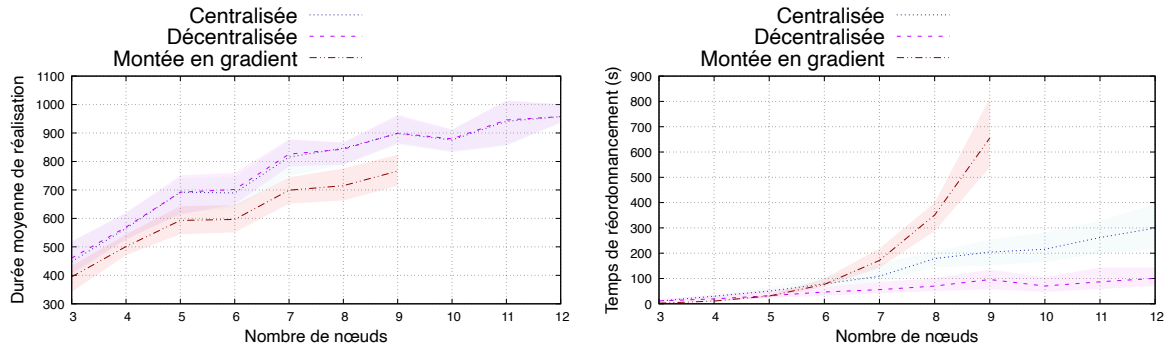
Les instances de MASTA+ que nous considérons sont telles que $m \in [2;12]$ nœuds/agents, $\ell \in [2;5]$ jobs et $n = 3 \times \ell \times m$ tâches. Il y a une ressource par tâche. Chaque ressource ρ_i est répliquée 3 fois et $|\rho_i| \in [0;100]$. Nous générons 10 instances de MASTA+, et pour chacune nous générons aléatoirement 10 allocations initiales. Les hypothèses que nous voulons tester sont : (1) le *flowtime* par notre stratégie est proche de celui obtenu par l'approche classique et (2) la décentralisation réduit significativement le temps d'ordonnancement.

Les figures 2a, 2b et 3 montrent les médianes et les déviations standards des métriques en fonction du nombre de nœuds avec $\ell = 4$ jobs. 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. Comme à chaque étape, l'algorithme de montée en gradient considère toutes les allocations possibles, le *flowtime* atteint est meilleur que celui de notre stratégie. Puisque le surcoût par rapport au *flowtime* de notre stratégie est de 25% vis-à-vis du gradient, notre stratégie de délégation semble être efficace même si le critère d'acceptabilité vis-à-vis du *makespan* nécessaire pour la convergence peut conduire à écarter certaines délégations susceptibles de réduire le *flowtime* (cf. section 4). Cela est dû au fait que la stratégie de délégation sélectionne les tâches distantes dont la délégation réduit le coût dans le but d'améliorer le taux de disponibilité locale qui est légèrement meilleur que celui atteint par l'algorithme de montée en gradient. De plus, étant donné que ce dernier évalue à chaque itération toutes les délégations possibles, son temps d'ordonnancement est bien supérieur à celui de notre stratégie de négociation. Par exemple, il est six fois plus élevé pour 9 agents et 4 jobs.

Il est intéressant de noter que l'écart entre les temps de réordonnancement des deux méthodes croît exponentiellement avec le nombre d'agents, tandis que l'écart pour le *flowtime* est globablement constant. 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 le *flowtime* par 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).

Enfin, on constate que l'efficacité de l'exécution de notre stratégie sur plusieurs cœurs croît avec le nombre d'agents et le nombre de jobs. Par exemple, avec un *flowtime* équivalent, la version décentralisée est 3 fois plus rapide pour 12 agents et 4 jobs que la version centralisée.



(a) Durée moyenne de réalisation des jobs

(b) Temps de réordonnancement

FIGURE 2 – *Flowtime* moyen et temps d'ordonnancement pour notre stratégie de négociation (centralisée ou décentralisée) et l'algorithme de montée en gradient.

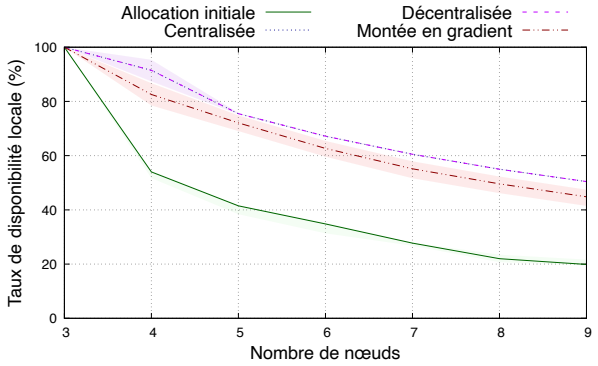


FIGURE 3 – Taux de disponibilité locale pour notre stratégie de négociation (centralisée et décentralisée), l'algorithme de montée en gradient et l'allocation initiale.

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 continue 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 donneur en choisissant un receveur qui n'est pas limitant pour les jobs impactés et en choisissant une tâche dont la délégation réduit le coût puisqu'elle sera exécutée localement. 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. Évidemment, 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.

Une étude comparative de notre stratégie avec différentes méthodes de résolution distribuées est en cours. Un certain nombre de nos expériences suggèrent que nous devons étendre notre cadre de négociation pour envisager (a) des échanges de tâches afin d'améliorer la durée moyenne de réalisation des allocations stables, et (b) une règle d'acceptabilité moins restrictive qui écarte actuellement certaines délégations qui pourraient réduire le *flowtime* moyen. En général, 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 remercions le comité de programme des JFSMA qui, par ses remarques, nous a permis d'améliorer cet article.

Références

[1] Bo An, Victor Lesser, David Irwin, and Michael Zink. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proc. of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 981–988, 2010.

[2] Quentin Baert, Anne-Cécile Caron, Maxime Morge, Jean-Christophe Routier, and Kostas Stathis. Stratégie situationnelle pour l'équilibrage de charge. In *Actes des 27ièmes journées francophones sur les systèmes multi-agents*, pages 9–18. Cépaduès, 2019.

[3] Soumya Banerjee and Joshua P. Hecker. A Multi-agent System Approach to Load-Balancing and Resource Allocation for Distributed Computing. In *Proc. of the 1st Complex Systems Digital Campus World E-Conference 2015*, pages 41–54. Springer International Publishing, 2017.

[4] Ellie Beauprez, Luc Bigand, Anne-Cécile Caron, Maxime Morge, and Jean-Christophe Routier. Réaffectation de tâches de la théorie à la pratique : état de l'art et retour d'expérience. In *Actes des 29ièmes journées francophones sur les systèmes multi-agents*, page 10. Cépaduès, 2021.

[5] Ellie Beauprez, Anne-Cécile Caron, Maxime Morge, and Jean-Christophe Routier. Stratégie multi-agents de négociation pour la réduction du flowtime. `https://tinyurl.com/MASTAPlus`, 2020. Rapport de recherche. Université de Lille.

[6] Ellie Beauprez, Anne-Cécile Caron, Maxime Morge, and Jean-Christophe Routier. A Multi-Agent Negotiation Strategy for Reducing the Flowtime. In *Proc. of 13th International Conference on Agents and Artificial Intelligence (ICAART)*, volume 1, pages 58–68, February 2021.

[7] Ellie Beauprez and Maxime Morge. Scala implementation of the Extended Multi-agents Situated Task Allocation. `https://gitlab.univ-lille.fr/maxime.morge/smastaplus`, 2020.

[8] Ulle Endriss, Nicolas Maudet, Fariba Sadri, and Francesca Toni. Negotiating Socially Optimal Allocations of Resources. *Journal of Artificial Intelligence Research*, 25 :315 – 348, 2006.

[9] Yichuan Jiang. A survey of task allocation and load balancing in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(2) :585–599, 2016.

[10] Yichuan Jiang and Zhaofeng Li. Locality-sensitive task allocation and load balancing in networked multiagent systems : Talent versus centrality. *Journal of Parallel and Distributed Computing*, 71(6) :822–836, 2011.

[11] Lightbend. Akka is the implementation of the actor model on the JVM. `http://akka.io`, 2020.

[12] Gauthier Picard. Optimisation sous contraintes distribuée : une introduction au domaine. In *Actes des 26ièmes journées francophones sur les systèmes multi-agents*, pages 43–52. Cépaduès, 2018.

[13] Michael L. Pinedo. *Scheduling. Theory, Algorithms, and Systems. Third Edition*. Springer, 2008.

[14] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1) :97–102, 1 1982.

[15] Andrea Schaerf, Y. Shoham, and Moshe Tennenholtz. Adaptive load balancing : A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2 :475–500, 1995.

[16] Oguz Selvitopi, Gunduz Vehbi Demirci, Ata Turk, and Cevdet Aykanat. Locality-aware and load-balanced static task scheduling for MapReduce. *Future Generation Computer Systems*, 90 :49–61, 2019.

[17] Joanna Turner, Qinggang Meng, Gerald Schaefer, and Andrea Soltoggio. Distributed Strategy Adaptation with a Prediction Function in Multi-Agent Task Allocation. In *Proc. of 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 739–747, 2018.

[18] Michael P Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1 :1–23, 1993.

[19] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion 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*, pages 15–28, 2012.