Un système multi-agent adaptatif pour la réallocation de tâches au sein d'un job MapReduce

Quentin Baert, Anne-Cécile Caron, Maxime Morge, Jean-Christophe Routier, Kostas Stathis

Quentin Baert, Anne-Cécile Caron, Maxime Morge,
Jean-Christophe Routier, Kostas Stathis

Un système multi-agent adaptatif pour la réallocation de tâches au sein d'un job MapReduce

Volume 3, nº 5-6 (2022), p. 557-585.

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 : pending

Revue Ouverte d'Intelligence Artificielle
Volume 3, n° 5-6, 2022, 557-585

Un système multi-agent adaptatif pour la réallocation de tâches au sein d'un job MapReduce

Quentin Baert[a], Anne-Cécile Caron[a], Maxime Morge[a], Jean-Christophe Routier[a], Kostas Stathis[b]

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

[b] Department of Computer Science, Royal Holloway, University of London, Egham TW20 0EX, UK
E-mail : Kostas.Stathis@rhul.ac.uk.

Résumé. — Nous étudions le problème de la réallocation de tâches pour l'équilibrage de *jobs* MapReduce dans des applications de traitement de données massives. Dans ce contexte, pour optimiser l'ordonnancement de tâches dans un seul *job* MapReduce, nous proposons une stratégie basée sur des agents coopératifs. L'originalité de notre stratégie réside dans la capacité des agents à identifier des opportunités au sein d'une allocation déséquilibrée qui leur permettent de déclencher des négociations « un-à-plusieurs » concurrentes entre agents dans le but de réallouer certaines tâches au sein du *job*. Notre contribution consiste à réallouer les tâches en fonction de la proximité des ressources pour qu'elles soient exécutées en tenant compte des capacités des nœuds sur lesquels sont situés les agents. Pour évaluer l'adaptivité et la réactivité de notre approche, nous avons implémenté un prototype et mené de nombreuses expériences dans un environnement hétérogène, selon différentes configurations. Cette large campagne d'expérimentations révèle que notre stratégie améliore significativement le temps d'exécution total par rapport au processus classique de Hadoop.

Mots-clés. — Système multi-agents, Résolution collective de problèmes, Négociation multi-agents.

1. Introduction

Les sciences des données nécessitent de traiter de grands volumes de données à l'aide d'un système de fichiers distribué et de la programmation parallèle. Ce traitement distribué de données pose de nouveaux défis pour l'allocation de tâches et l'équilibrage de charge. En particulier il nécessite d'améliorer la réactivité de ces systèmes aux changements ainsi que leur adaptabilité aux nombreuses configurations sans avoir recours à une expertise de l'utilisateur.

Cet article s'intéresse à la classe d'applications où (a) les ressources (*e.g.* les données) nécessaires à la réalisation d'une tâche sont distribuées sur les différents

nœuds de calcul, (b) certains de ces nœuds peuvent être sujets à des aléas d'exécution, *e.g.* un ralentissement ou une latence dans la communication, et (c) le nombre de tâches interdit l'utilisation d'une approche centralisée pour calculer l'allocation. Dans la mesure où plusieurs ressources sont nécessaires pour réaliser une tâche, toute allocation implique que certaines de ces ressources doivent être collectées auprès des autres nœuds, ce qui provoque un surcoût du temps d'exécution de la tâche [24]. Dans cette classe d'application, l'allocation de tâches peut-être remise en cause au cours de l'exécution en prenant en compte la distribution des ressources au sein du système.

Parmi cette classe d'applications pratiques, nous considérons ici MapReduce [17] qui est le modèle de calcul distribué prépondérant pour le traitement d'un volume très important de données sur un cluster. Les tâches sont divisées en deux ensembles de tâches *map* et *reduce*, qui sont distribuées sur les nœuds. L'allocation des tâches *reduce* entre les nœuds est fixée *a priori* par une fonction de partitionnement. Par exemple, dans Hadoop [44], qui est l'implémentation la plus connue, le partitionnement par défaut calcule la valeur de hachage de la clef de la tâche modulo le nombre de *reducers*. Cette allocation peut poser des problèmes. Premièrement, de nombreux biais sur les données (cf. [30, 31]) peuvent déséquilibrer la charge de travail au cours de la phase de *reduce*. Deuxièmement, des performances hétérogènes des nœuds vont amener à une allocation non équitable. Troisièmement, des variations brutales de performances dues à des facteurs exogènes peuvent remettre en cause l'équilibre de la charge de travail.

Afin d'aborder le problème d'équilibrage de charge et d'allocation des tâches dans des applications telles que celles qui motivent ce travail, les technologies multi-agents ont fait l'objet d'une grande attention [1, 12]. La plupart des travaux existants adoptent une approche basée sur des enchères [42, 45, 46]. Ils modélisent le problème de l'équilibrage de charge comme un jeu non-coopératif dans le but d'optimiser des métriques locales à l'utilisateur, plutôt que des métriques globales du système comme le temps d'exécution global considéré dans cet article. Nous supposons que les agents sont coopératifs. Ils partagent le même objectif : minimiser le temps d'exécution global. Nous supposons également qu'il n'y a pas de connaissances partagées, notamment pas de vision globale de l'allocation de tâches. Cependant, les agents ont un modèle de leurs pairs, c'est-à-dire qu'ils sont capables de calculer le coût d'une tâche pour leurs pairs. Enfin, nous supposons, conformément à notre application pratique, qu'une tâche peut être exécutée par n'importe quel agent, sans préemption, ni ordre de précédence. Une tâche est indivisible, sans date butoir et non partageable, c'est-à-dire qu'à un instant donné, une tâche appartient à un seul agent.

Dans cet article, nous formalisons le problème multi-agents de l'allocation de tâches situées. Nous proposons un processus continu et dynamique de réallocation de tâches qui se déroule en concurrence avec l'exécution des tâches. Ainsi, le système distribué est adaptatif aux phénomènes disruptifs (*e.g.* le ralentissement de nœuds). Quand les agents identifient localement des opportunités au sein d'une allocation déséquilibrée, ils déclenchent des négociations « un-à-plusieurs » concurrentes pour réallouer certaines tâches. En plus de la décentralisation qui évite les problèmes de performance liés à un contrôle global, nous montrons ici qu'une approche multi-agents

de l'allocation de tâches situées favorise deux exigences cruciales (a) la concurrence – la réallocation et l'exécution des tâches sont concurrentes, et (b) l'adaptation – une réallocation est déclenchée quand un événement disruptif se produit. Cet article est une extension significative de nos travaux précédents. À la différence de [7], nous ne supposons pas que chaque tâche a un coût intrinsèque, mais dépend de la localisation des ressources. En complément de [9], cet article est une traduction de [10] où nous introduisons un processus multi-enchères qui permet à un agent de participer à plusieurs négociations concurrentes dans le but d'améliorer l'efficacité du système. De plus, nous présentons de nombreuses expériences pour évaluer empiriquement (a) l'adaptabilité de notre système multi-agents dans un environnement hétérogène, (b) la réactivité du système multi-agents obtenue grâce au processus multi-enchères, et (c) l'adéquation de la stratégie de l'agent par rapport à l'infrastructure sous-jacente.

Plus spécifiquement, nos contributions sont les suivantes :

- nous formalisons le problème multi-agents de l'allocation de tâches situées, dans lequel les coûts des tâches diffèrent d'un agent à l'autre selon la localisation des ressources ;
- nous concevons une version multi-agents du modèle MapReduce, dans la configuration d'un système distribué qui résout le problème du biais de partitionnement des données. Le processus de réallocation de tâches qui s'appuie sur des négociations concurrentes entre agents se déroule tout au long du traitement du *job* MapReduce pour réagir à un environnement en perpétuelle évolution ;
- nous menons de nombreuses expériences sur des données réelles. Les résultats expérimentaux montrent que notre système améliore le temps d'exécution grâce à un surcoût de calcul négligeable et qu'il s'adapte à un environnement de calcul hétérogène.

Cet article s'organise ainsi. La section 2 présente des travaux connexes. La section 3 définit la délégation socialement rationnelle de tâches, ce qui permet aux agents d'améliorer localement une allocation de tâches. La section 4 illustre le processus de négociations concurrent à l'exécution des tâches. La section 5 décrit les stratégies, c'est-à-dire comment les agents choisissent quelle tâche exécuter/négocier. Notre application pratique et notre validation empirique sont décrites dans la section 6. Enfin, la section 7 résume notre contribution et présente nos perspectives.

2. Travaux connexes

Le contexte de cet article est celui d'un unique *job* MapReduce : le problème de l'ordonnancement des tâches consiste en l'affectation de tâches *map* et *reduce* comme suggéré par Selvitopi et al. in [41]. Ce problème ne doit pas être confondu avec celui de l'ordonnancement des *jobs*, où il est question de l'allocation et de l'utilisation des ressources dans le cas de plusieurs *jobs* MapReduce comme discuté par Banerjee et Hecker dans [11]. Plusieurs biais sur les données dans les applications MapReduce sont identifiés dans [30, 31]. Dans cet article, nous nous focalisons sur le biais de

partitionnement qui engendre un déséquilibre de l'allocation des tâches de *reduce* entre les nœuds, et qui ralentit ainsi la phase de *reduce* pénalisée par le *reducer* le plus chargé. Ce biais est traité dans [16, 34] où la solution repose sur un paramétrage qui nécessite une connaissance préalable des données et une expertise de l'environnement de calcul distribué. Dans [20], chaque tâche *reduce* est affectée au nœud qui est son « centre de gravité » en prenant en compte le biais de partitionnement et la localité des données. Dans cet article, nous répondons au biais de partitionnement grâce à des réallocations dynamiques et adaptatives de tâches, qui sont concurrentes avec la consommation des tâches. Ainsi, la réallocation s'adapte au traitement des données. Cela nous permet de prendre en compte les problèmes réels suivants : (a) l'absence de connaissance préalable sur les données, (b) l'estimation inexacte du temps d'exécution des tâches, et (c) les aléas d'exécutions (chute de performance d'un nœud, latence réseau). À notre connaissance, aucune autre proposition n'est, comme la nôtre, réactive et capable de passer à l'échelle.

Nous proposons ici une comparaison de notre travail avec les méthodes existantes les plus significatives pour l'allocation de tâches et l'équilibrage de charge, qui sont récapitulées dans la table 2.1. Cette grille d'analyse classifie ces travaux selon les aspects essentiels exigés par notre application pratique : le déploiement du schéma de conception MapReduce pour le traitement de données massives.

Les problèmes d'ordonnancement classiques ont fait l'objet d'un grand nombre d'études et de recherches. Ces dernières ont abouti à des ordonnanceurs hors-ligne pour des modèles simples [15]. Le problème de la minimisation du *makespan* (le temps auquel la dernière tâche est achevée) pour n tâches sur m machines hétérogènes (avec des capacités différentes), appelé $R||C_{max}$, est NP-difficile [22]. Les algorithmes pseudo-polynomiaux conçus pour ce problème incluent : l'heuristique *earliest completion time* [23] (ECT), les heuristiques biphasées basées sur la programmation linéaire [32], les heuristiques de recherche locale [21] et l'algorithme *branch and bound* [35]. Ni ces algorithmes centralisés, ni les plus récents [37], ni même l'heuristique d'approximation ECT qui atteint des résultats acceptables avec un coût computationnel faible, ne peuvent être appliqués à notre scénario où les tâches sont nombreuses (par exemple 100 000 tâches dans la section 6). La stratégie présentée section 5 est une heuristique de recherche locale décentralisée.

L'ordonnancement multi-agents [24] a suscité beaucoup d'intérêt dans le cadre du problème d'équilibrage de charge pour les systèmes distribués. Ce problème est différent des problèmes d'ordonnancement classiques en raison des aspects suivants :

- **Décentralisation.** Un contrôle global constitue un goulot d'étranglement pour les performances puisqu'il nécessite de collecter les informations sur l'ensemble du système en temps réel. À la place, l'allocation de tâches peut être négociée par des agents qui représentent les nœuds. Jiang et al. proposent dans [28] un mécanisme de négociation d'allocation basé sur la réputation, pour l'équilibrage de charge afin de réduire le temps d'accès aux ressources et donc le temps de réponse.

- **Adaptation.** les problèmes d'ordonnancement classiques sont statiques. Cependant, une estimation incorrecte du temps d'exécution des tâches et des phénomènes disruptifs (consommation de tâches, ralentissement de nœuds, etc.) peuvent nécessiter des modifications conséquentes de l'allocation pour conserver l'optimalité. Dans [45], Turner et al. combinent un apprentissage supervisé d'une classification avec un processus interne de prise de décision pour l'affectation de tâches. Schaerf et al. explorent dans [39] le comportement adaptatif d'agents pour un équilibrage de charge efficace en utilisant un apprentissage par renforcement multi-agents. Ces méthodes ne peuvent pas être utilisées pour notre classe d'applications pratiques, car aucun schéma prédictif généralisable ni aucun modèle préalable sur les données ou sur l'environnement ne sont disponibles.

Il existe de nombreuses études connexes (voir [24] pour un état de l'art récent) qui se distinguent les unes des autres selon les aspects suivants :

- **Objectifs.** La minimisation du *makespan* est l'objectif d'optimisation le plus largement considéré pour l'allocation de tâches. Selvitopi et al. [41] proposent un ordonnancement de tâches pour minimiser le *makespan*, cependant l'ordonnancement de tâches est réalisé une fois pour toute avant la phase de *reduce* alors que notre méthode remet en cause cette allocation tout au long du processus. La minimisation du temps moyen ou global de l'ensemble des tâches correspond à minimiser le temps d'attente moyen des tâches. Le rendement mesure le nombre de tâches réalisées par unité de temps. La fiabilité mesure la probabilité qu'une tâche soit exécutée avec succès. Attiya et Hamam distinguent dans [4] la fiabilité relative aux nœuds, c'est-à-dire la fiabilité des ressources/calcul et la fiabilité des communications.
- **Distribution.** Les ressources sont localisées sur les nœuds. Elles sont accessibles ou peuvent être partagées pour exécuter les tâches. Il faut donc prendre en considération la localisation des agents et l'accessibilité des ressources. Dans [26], Jiang et Jiang considèrent que le nombre de tâches allouées à un nœud est proportionnel à ses ressources et aux ressources de ses accointances. Jiang et Li proposent dans [27] un modèle d'allocation de ressources sensible à la localité. Ce modèle prend en compte la distance entre les nœuds et la localisation des ressources. Jiang et Zhichuan présentent dans [25] un mécanisme d'allocation basé sur des ressources contextuelles : si un nœud a une plus grande expérience pour l'exécution des tâches, alors ce nœud peut avoir un meilleur accès aux ressources. Dans cet article, les nœuds sont équidistants les uns des autres tant dans le réseau d'accointances que dans le réseau physique. L'efficacité d'une allocation de tâches dépend à la fois de la proportion des ressources (données) qui sont locales et des performances intrinsèques (coûts).
- **Coopération.** Garg et al. distinguent dans [19] les méta-scheduler qui optimisent les métriques centrées sur le système et les approches plus récentes qui se focalisent sur les métriques centrées sur l'utilisateur. La plupart des

travaux existants adoptent une approche fondée sur le marché : ils modélisent le problème d'équlibrage des charges comme un jeu non-coopératif. Walsh et Wellman présentent dans [46] un protocole d'allocation de tâches entre agents qui acquièrent et fournissent des biens pour le compte de consommateurs et de producteurs. Kraus et al. considèrent dans [29] que, même si chaque agent tente de maximiser son bénéfice, les agents doivent coopérer pour réaliser les tâches. An et al. proposent dans [3] un mécanisme de négociation distribuée où des agents égoïstes négocient à la fois un prix de contrat et une pénalité de désengagement. Penmatsa et Chronopoulos formulent dans [38] le problème d'équilibrage de charges comme un jeu non-coopératif entre les utilisateurs qui essaient de minimiser le temps de réponse attendu pour leur propres tâches. Un nombre plus restreint de travaux s'appuient sur des agents coopératifs qui négocient pour optimiser des métriques centrées sur le système, telles que le temps d'achèvement global que nous utilisons dans cet article. Par exemple, Shehory and Kraus considèrent dans [42] l'affectation de tâches à des coalitions d'agents car ils ne peuvent pas réaliser les tâches seuls. À la différence, dans cet article, nous supposons qu'une tâche, qui peut être exécutée par n'importe quel agent seul sans préemption ni contrainte de précédence, est indivisible, non partageable (*i.e.* une tâche n'appartient qu'à un seul agent à la fois) et sans date butoir.

	Décentralisation	**Adaptation**	**Coopération**	**Distribution**	**Objectifs**
Ibarra and Kim [23]	-	-	-	-	minimisation du *Makespan*
Lenstra et al. [32]	-	-	-	-	minimisation du *Makespan*
Hariri and Potts [21]	-	-	-	-	minimisation du *Makespan*
Martello et al. [35]	-	-	-	-	minimisation du *Makespan*
Jiang [28]	✓	✓	-	✓	temps de réponse
Selvitopi et al. [41]	✓	-	-	✓	minimisation du *Makespan*
Turner et al. [45]	✓	✓	✓	-	rendement
Schaerf et al. [39]	✓	✓	✓	✓	rendement
Jiang and Jiang [26]	✓	✓	✓	✓	rendement
Jiang and Li [27]	✓	✓	✓	✓	minimisation du temps de réponse
Jiang and Zhichuan [25]	✓	✓	✓	✓	minimisation du temps de réponse
Walsh and Wellman [46]	✓	✓	-	✓	une allocation
Kraus et al. [29]	✓	✓	-	-	une allocation
An et al. [3]	✓	✓	-	✓	minimisation du temps de réponse
Penmatsa and Chronopoulos [38]	✓	-	-	-	minimisation du temps de réponse
Shehory and Kraus [42]	✓	✓	✓	-	minimisation du *Makespan*
MAS4Data	✓	✓	✓	✓	minimisation du *Makespan*

TABLE 2.1 – Grille d'analyse des travaux connexes suivant les principaux aspects.

D'une manière générale, les problèmes d'ordonnancement où les tâches sont réalisées en parallèle par plusieurs exécutants se distinguent non seulement de par les objectifs visés mais également de par leurs ingrédients (ressources, tâches, exécutants) [13]. Nous considérons ici que les ressources sont transférables (*i.e.* utilisables par un autre exécutant que celui qui la possède initialement) et duplicables. Les tâches

sont mono-exécutant car chacune d'elles est affectée à un seul exécutant. Les exécutants sont multi-tâches car chacun réalise séquentiellement plusieurs tâches. Leurs efficacités dans la réalisation des tâches sont hétérogènes.

3. Allocation de tâches situées

Nous formalisons maintenant le problème de l'allocation multi-agents de tâches situées (MASTA). Les tâches ont des coûts différents selon les agents, en fonction de la localité des ressources.

Définition 3.1 (MASTA). — *Un problème d'allocation multi-agents de tâches situées de taille* (k, m, n) *avec* $k \geqslant 1$, $m \geqslant 2$ *et* $n \geqslant 1$ *est un n-uplet MASTA* $= \langle Node, \mathcal{A}, \mathcal{T}, l, d, c\rangle$ *tel que :*

- $Node = \{node_1, \ldots, node_k\}$ *est un ensemble de k nœuds ;*
- $\mathcal{A} = \{1, \ldots, m\}$ *est un ensemble de m agents totalement connectés entre eux ;*
- $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *est un ensemble de n tâches à traiter ;*
- $l : \mathcal{A} \mapsto Node$ *donne la localisation d'un agent ;*
- $d : \mathcal{T} \times Node \mapsto \mathbb{N}^+$ *donne le nombre de ressources d'une tâches* τ *situées sur un nœud x ;*
- $c : \mathcal{T} \times Node \mapsto \mathbb{R}_+^*$ *donne le coût d'une tâche* τ *lorsqu'elle est exécutée sur un nœud de telle sorte que plus les ressources nécessaires sont locales, moins elle coûte chère :*

$$\forall i, j \in \mathcal{A}, d(\tau, l(i)) > d(\tau, l(j)) \Rightarrow c(\tau, l(i)) \leqslant c(\tau, l(j)) \tag{3.1}$$

Dans la suite de l'article, on écrira l_i (resp. $c_i(\tau)$, $d_i(\tau)$) à la place de $l(i)$ (resp. de $c(\tau, l_i)$, $d(\tau, l_i)$). De la même façon, on écrira d_τ pour $\Sigma_{node \in Node} d(\tau, node)$. On dit que τ est locale (resp. semi-locale, distante) pour l'agent i si $d_i(\tau) = d_\tau$ (resp. $d_i(\tau) < d_\tau$, $d_i(\tau) = 0$).

Dans la suite de cette section, nous considérons un problème MASTA particulier. Nous évaluons d'un point de vue collectif une allocation de tâches répondant à ce problème, en considérant le temps nécessaire pour achever la dernière tâche, *i.e.* le *makespan*.

Définition 3.2 (Allocation de tâches/Charge de travail/Makespan). — *Une allocation de tâches P est une partition des tâches parmi les agents, i.e un ensemble de m lots de tâches* $\{P(1), \ldots, P(m)\}$ *tel que :*

$$\cup_{i \in \mathcal{A}} P(i) = \mathcal{T} \tag{3.2}$$

$$\forall i \in \mathcal{A}, \forall j \in \mathcal{A} \setminus \{i\}, P(i) \cap P(j) = \emptyset \tag{3.3}$$

La charge de travail de l'agent $i \in \mathcal{A}$ *pour l'allocation P est définie par :*

$$w_i(P) = \Sigma_{\tau \in P(i)} c_i(\tau) \tag{3.4}$$

Le makespan de P est défini par :

$$C_{max}(P) = \max\{w_i(P) \mid i \in \mathcal{A}\} \tag{3.5}$$

Voici un exemple pour illustrer les différentes définitions.

Exemple 3.3. — Considérons le problème $MASTA^{ex} =< Node, \mathcal{A}, \mathcal{T}, l, d, c >$ de taille $(2, 2, 7)$, où $Node = \{node_1, node_2\}$, $\mathcal{A} = \{1, 2\}$ et $\mathcal{T} = \{\tau_1, \ldots, \tau_7\}$ avec $l(1) = node_1$, $l(2) = node_2$. Les localisations et coûts des tâches sont donnés dans le tableau 3.1.

TABLE 3.1 – Localisations et coûts des tâches

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
$d_1(\tau_k)$	1	0	3	6	1	6	0
$d_2(\tau_k)$	0	4	6	12	4	1	7
$c_1(\tau_k)$	1	8	15	30	9	8	14
$c_2(\tau_k)$	2	4	12	24	6	13	7

Soient les allocations Pmks et P telles que Pmks $= \{\{\tau_1, \tau_3, \tau_5, \tau_6\}, \{\tau_2, \tau_4, \tau_7\}\}$ et $P = \{\{\tau_2, \tau_4, \tau_6\}, \{\tau_1, \tau_3, \tau_5, \tau_7\}\}$. Pmks est optimale puisque $C_{max}(\text{Pmks}) = 35$ ($w_1(\text{Pmks}) = 33$ et $w_2(\text{Pmks}) = 35$). Comme montré en haut de la figure 3.1, ce n'est pas le cas de P puisque $w_1(P) = 46$, $w_2(P) = 27$, et donc $C_{max}(P) = 46$.

Les agents modifient l'allocation de tâches en négociant des délégations de tâches. La délégation δ d'une tâche τ de l'agent i à l'agent j a pour objectif l'amélioration du *makespan*, c'est-à-dire l'équilibre des charges de travail, entre les deux agents. La définition suivante formalise les conditions nécessaires à une telle délégation.

DÉFINITION 3.4 (Délégation socialement rationnelle). — *Soit P une allocation de tâches. La délégation δ de la tâche τ par l'agent i à l'agent j permet d'obtenir l'allocation $\delta(P) = \{P'(1), \ldots, P'(m)\}$ telle que :*

$$\forall k \in \mathcal{A} \setminus \{i, j\}, P'(k) = P(k) \tag{3.6}$$

$$P'(i) = P(i) \setminus \{\tau\} \wedge P'(j) = P(j) \cup \{\tau\} \tag{3.7}$$

La délégation est socialement rationnelle si et seulement si :

$$w_j(P) + c_j(\tau) < w_i(P) \tag{3.8}$$

On peut noter qu'une délégation socialement rationnelle est un cas particulier d'échange équitable (*equitable deal*) défini dans [18] qui est restreint à deux agents, une seule tâche et où la fonction d'utilité considérée est la charge de travail des agents $w_i(P)$.

Comme la délégation socialement rationnelle δ décroît strictement le *makespan* local entre deux agents, elle ne peut pas augmenter le *makespan* global ($C_{max}\big(\delta(P)\big) \leqslant C_{max}(P)$).

Nous pouvons maintenant noter $\Gamma_i(P)$ l'ensemble des délégations socialement rationnelles pour l'agent i :

$$\Gamma_i(P) = \{\tau \in P(i) \mid \exists j \in \mathcal{A} \setminus \{j\}, w_j(P) + c_j(\tau) < w_i(P)\} \tag{3.9}$$

Une allocation de tâches P est dite stable si aucun agent ne peut réaliser de délégation socialement rationnelle, *i.e.* $\forall i \in \mathcal{A}, \Gamma_i(P) = \emptyset$.

Exemple 3.5. — Soit δ_1 la délégation de la tâche τ_6 de l'agent 1 vers l'agent 2 dans l'exemple 3.3, elle produit l'allocation de tâches $P' = \delta_1(P)$. Comme $w_1(P') = 38$ et $w_2(P') = 40$, δ_1 améliore le makespan (*i.e.* $C_{max}(P') = 40$). Cependant, P' n'est pas stable.

Soit $P'' = \delta_2(P')$ l'allocation obtenue via la délégation δ_2 de la tâche τ_1 par l'agent 2 à l'agent 1. Même si P'' n'est pas optimale ($C_{max}(P'') > C_{max}(\text{Pmks})$), P'' est stable. La figure 3.1 présente les charges des agents après les délégations δ_1 et δ_2.

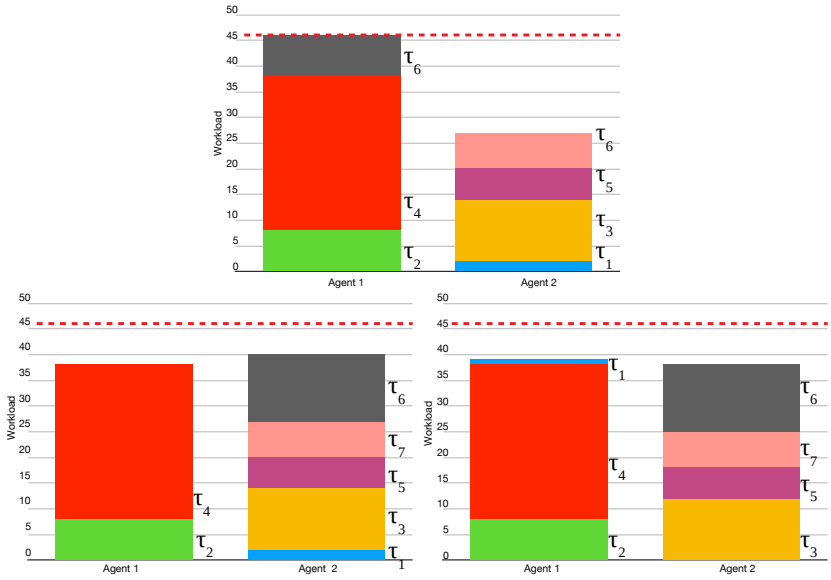


FIGURE 3.1 – Les charges de travail des agents 1 et 2 pour l'allocation P (en haut), P' (en bas à gauche) et P'' (en bas à droite) pour l'exemple fil-rouge.

Il est intéressant de noter que le processus d'équilibrage des charges de travail qui consiste en une séquence de délégations socialement rationnelles est fini.

PROPOSITION 3.6. — *Quelque soit P une allocation, toute séquence de délégations socialement rationnelles issue de P est finie car elle mène à une allocation stable.*

Démonstration. — Comme il y a un nombre fini d'allocations, la preuve est identique à celle du lemme 3 dans [18]. □

4. PROCESSUS DE NÉGOCIATION

Cette section présente le processus itéré de négociation qui est concurrent à la consommation des tâches. Quand une tâche est exécutée, elle est retirée de l'ensemble

des tâches et le système multi-agents cherche à minimiser le *makespan* du nouveau problème MASTA obtenu.

L'exécution d'une tâche est un évènement disruptif qui modifie les paramètres du problème et l'allocation des tâches.

Définition 4.1 (Consommation de tâche). — *Soit P l'allocation de tâches courante pour le problème MASTA $= \langle Node, \mathcal{A}, \mathcal{T}, l, d, c\rangle$. La consommation γ de la tâche τ par l'agent i aboutit à l'allocation $P' = \gamma(P)$ pour le problème MASTA$' = \langle Node, \mathcal{A}, \mathcal{T}', l, d, c\rangle$ telle que :*

$$\mathcal{T}' = \mathcal{T} \setminus \{\tau\} \tag{4.1}$$

$$P'(i) = P(i) \setminus \{\tau\} \tag{4.2}$$

$$\forall j \in \mathcal{A} \setminus \{i\}, P'(j) = P(j) \tag{4.3}$$

La séquence des consommations de tâches retire toutes les tâches de l'allocation initiale jusqu'à l'allocation vide $\perp$. De manière évidente la consommation d'une tâche peut entraîner une diminution du *makespan*.

Processus décentralisé de délégations de tâches. Pour déléguer des tâches, les agents réalisent des négociations concurrentes. Chaque négociation est basée sur le protocole Contract Net [43]. Il y a 3 étapes de décision durant une négociation : (a) le choix de la tâche à négocier par la stratégie de l'initiateur décrite dans la section 5, (b) les propositions/refus des pairs qui vérifient si la délégation est socialement rationnelle ou non, et (c) la sélection par l'initiateur du vainqueur de l'enchère. Nous considérons ici que l'initiateur choisit l'enchérisseur ayant la plus petite charge de travail afin de diminuer le *makespan*.

Comme il n'y a pas de partage de connaissances, un agent a des croyances partielles, éventuellement erronées, concernant l'allocation courante P. En effet, l'agent i connaît sa charge de travail $w_i(P)$ et dispose de sa propre base de croyances :

$$\mathcal{B}_i(P) = \langle w_1^i(P), \ldots, w_{i-1}^i(P), w_{i+1}^i(P), \ldots, w_m^i(P)\rangle \tag{4.4}$$

où $w_j^i(P)$ est ce que l'agent i croit connaître de la charge de travail de l'agent j dans l'allocation P. C'est à partir de cette base de croyances que l'agent décide s'il doit ou non être à l'initiative d'une délégation. Cette base de croyances lui permet en effet d'identifier l'ensemble des délégations qui sont potentiellement socialement rationnelles $\Gamma_i^{\mathcal{B}}(P)$ et que l'agent i peut initier à partir de l'allocation P. Formellement,

$$\Gamma_i^{\mathcal{B}}(P) = \{\tau \in P(i) \mid w_j^i(P) + c_j(\tau) < w_i(P)\}. \tag{4.5}$$

Si $\Gamma_i^{\mathcal{B}}(P) = \emptyset$, alors l'agent n'initie pas de négociation. Le calcul du *makespan* local s'appuie sur sa base de croyances. Comme le système est décentralisé, cette base est éventuellement erronée. Cependant, un agent informe tous les autres agents de sa charge de travail au début du traitement (*i.e.* lorsqu'il se fait connaître de ses pairs), lorsqu'il envoie des messages en tant qu'initiateur ou enchérisseur durant la négociation, et à chaque fois qu'il consomme une tâche. Ainsi la base de croyances

d'un agent est mise à jour quand il reçoit un appel d'offre, et l'agent refusera une délégation de tâche qui n'est pas socialement rationnelle. Une négociation réussie aboutit nécessairement à une délégation de tâche socialement rationnelle qui ne peut pas détériorer le *makespan*. Quand un agent identifie, selon sa base de croyances, des opportunités dans une allocation déséquilibrée, il initie une négociation. Ces décisions locales favorisent l'adaptabilité du système multi-agents.

Consommations et délégations concurrentes. Les délégations et consommations de tâches sont concurrentes et complémentaires, puisque la disparition d'une tâche peut permettre de nouvelles délégations socialement rationnelles. La figure 4.1 représente leur impact sur l'allocation, jusqu'à ce que toutes les tâches soient exécutées. À partir de l'allocation initiale P_0, les agents réalisent des délégations socialement rationnelles pour améliorer le *makespan* (*e.g.* le chemin de P_0 à P_k) jusqu'à la consommation d'une tâche (*e.g.* l'arc de P_k à P'_0), ce qui interrompt un chemin vers une allocation stable (*e.g.* le chemin en gris de P_k à P_{stable}). Une consommation de tâche peut aussi se produire après avoir atteint une allocation stable (par ex. après P'_{stable}) ou non (par ex. P_k).

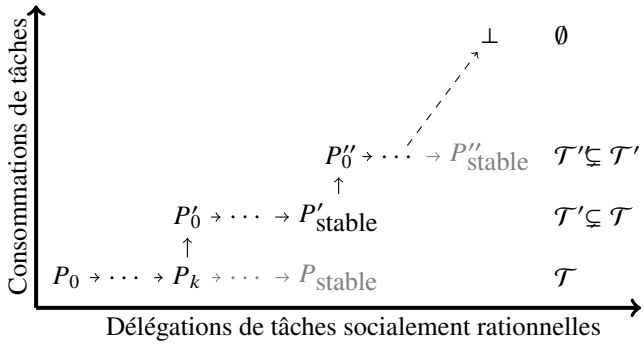


Figure 4.1 – Consommations de tâches (arcs verticaux) et délégations de tâches (arcs horizontaux) concurrentes.

Enchères multiples. Même si un agent ne peut pas être en même temps initiateur d'une négociation et enchérisseur dans une autre, plusieurs négociations qui impliquent différents groupes d'agents peuvent avoir lieu en concurrence. Les travaux précédents [6] s'appuyaient sur un mécanisme d'enchères uniques, dans lequel un agent ne pouvait être engagé en tant qu'enchérisseur que dans une seule négociation à la fois. En complément, nous présentons ici un processus multi-enchères qui permet à un agent d'être enchérisseur dans plusieurs négociations concurrentes, comme dans [2], afin d'améliorer la réactivité du système. Grâce à cela, comme établi par nos résultats expérimentaux dans la section 6, l'écart des charges entre le nœud le plus chargé et le moins chargé est comblé plus rapidement et le processus d'équilibrage est plus rapide.

Afin de contrer le problème de l'enchérisseur enthousiaste [40], nous adoptons une approche conservative qui garantit que les délégations de tâches sont socialement rationnelles. Dans cet objectif, un enchérisseur calcule une *surcharge*.

Définition 4.2 (*surcharge*). — *Soit P une allocation, $\mathcal{D} \subset \mathcal{T}$ les tâches qui sont actuellement négociées et $\mathcal{D}_j$ l'ensemble des tâches pour lesquelles l'agent j a déjà fait une proposition. La surcharge de l'agent j est :*

$$v_j(\mathcal{D}) = \Sigma_{\tau \in \mathcal{D}_j} c_j(\tau) \tag{4.6}$$

La charge potentielle pour l'agent j, qui représente sa charge de travail s'il gagne toutes les enchères dans lesquelles il est impliqué ($w_j(P) + v_j(\mathcal{D})$), permet à un enchérisseur de ne pas être trop optimiste et de ne faire que des propositions qui déboucheront nécessairement sur une délégation socialement rationnelle. Dans le but d'évaluer la délégation d'une tâche τ de l'agent i, l'enchérisseur j adopte la stratégie suivante :

- soit $w_j(P) + c_j(\tau) \geqslant w_i(P)$, alors l'enchérisseur décline la délégation qui n'est pas socialement rationnelle ;
- soit $w_j(P) + c_j(\tau) < w_i(P) \leqslant w_j(P) + v_j(\mathcal{D}) + c_j(\tau)$, alors l'enchérisseur retarde l'évaluation de la délégation qui dépend des négociations en cours ;
- soit $w_j(P) + v_j(\mathcal{D}) + c_j(\tau) < w_i(P)$, alors l'enchérisseur fait une proposition puisque la délégation de tâche est socialement rationnelle quelque soit les issues des négociations en cours. Donc,

  $$v_j(\mathcal{D} \cup \{\tau\}) = v_j(\mathcal{D}) + c_j(\tau) \tag{4.7}$$

  De plus, l'enchérisseur informe l'initiateur de sa charge de travail potentiel.

Dans le dernier cas, quand la négociation se termine :

- soit l'enchérisseur est sélectionné ($P'_j = P_j \cup \{\tau\}$) et sa charge de travail est mise à jour $w_j(P') = w_j(P) + c_j(\tau)$;
- soit ce n'est pas le cas ($P'_j = P_j$), sa charge de travail reste la même $w_j(P') = w_j(P)$.

Dans les deux cas, la *surcharge* est mise à jour :

$$v_j(\mathcal{D}) = v_j(\mathcal{D} \cup \{\tau\}) - c_j(\tau) \tag{4.8}$$

Finalement, les délégations en cours sont réévaluées.

Exemple 4.3 (Multi-enchères). — Soit P une allocation entre les agents $\mathcal{A} = \{1, 2, 3, 4, 5, 6\}$ telle que les charges de travail sont :

$$w_1(P) = 30 \quad w_2(P) = 45 \quad w_3(P) = 39$$
$$w_4(P) = 28 \quad w_5(P) = 34 \quad w_6(P) = 40$$

Nous nous focalisons ici sur la *surcharge* de l'agent 1 et son influence sur les négociations. Nous supposons que les agents 2, 3, 4, 5 et 6 demandent les délégations des tâches τ_2, τ_3, τ_4, τ_5 and τ_6 respectivement. Le coût de ces tâches pour l'agent 1 sont :

$$c_1(\tau_2) = 6 \quad c_1(\tau_3) = 2 \quad c_1(\tau_4) = 4 \quad c_1(\tau_5) = 3 \quad c_1(\tau_6) = 3$$

La figure 4.2 illustre l'évolution de la *surcharge* de l'agent 1 au cours de ses interactions avec ses pairs. L'agent 1 peut faire une proposition aux agents 2 et 3 (messages 2 et 4). Cependant, la délégation suggérée par l'agent 4 (message 5), qui n'est pas socialement rationnelle, est déclinée (message 6). L'agent 1 doit retarder l'évaluation des délégations des agents 5 et 6 (messages 7 et 8) qui dépendent des négociations en cours. Quand la délégation de la tâche τ_2 est confirmée (message 10), l'agent 1 peut décliner la délégation de τ_5 (message 11) mais la rationalité de la délégation de τ_6 ne peut être ni exclue ni confirmée. Enfin, quand la délégation de la tâche τ_3 est rejetée (message 12), l'agent 1 peut faire une proposition pour τ_6 (message 12).

5. Stratégies

Dans la mesure où les délégations et consommations de tâches sont concurrentes, la stratégie d'un agent doit sélectionner la prochaine tâche à exécuter ou à déléguer.

Définition 5.1 (Stratégie). — *Soit P une allocation, la stratégie d'un agent i est le couple $(perform_i, negotiate_i)$ où :*

- $perform_i : P(i) \mapsto \mathcal{T} \cup \{\perp\}$*, sélectionne la prochaine tâche à réaliser ou aucune (notée $\perp$) si $P(i) = \emptyset$;*
- $negotiate_i : \Gamma_i^{\mathcal{B}}(P) \mapsto \mathcal{T} \cup \{\perp\}$*, sélectionne la prochaine tâche à négocier ou aucune $\Gamma_i^{\mathcal{B}}(P) = \emptyset$.*

Précisons que la tâche que la stratégie sélectionne pour négocier doit correspondre à une délégation socialement rationnelle. Dans la suite, nous proposons deux stratégies. Tandis que la stratégie agnostique vis-à-vis de la localité se base uniquement sur la fonction de coût, la stratégie situationnelle prend en compte la localisation des ressources.

Stratégie agnostique vis-à-vis de la localité. Cette stratégie amène un agent à exécuter les tâches les plus petites de son propre lot et à négocier les plus grandes parmi celles qui permettent des délégations potentiellement socialement rationnelles. Dans ce but, cette stratégie nécessite simplement que l'agent trie son lot de tâches selon leurs coûts. On peut remarquer que cette stratégie prend implicitement en compte le temps nécessaire à la récupération d'une ressource puisqu'elle favorise l'exécution des tâches qui devraient être plus coûteuses pour les pairs et la délégation des tâches dont les ressources sont distantes et devraient donc être moins coûteuses pour les pairs. Dans la seconde stratégie cette prise en compte est explicite.

Stratégie situationnelle. Selon cette stratégie, un agent exécute d'abord les grandes tâches locales et négocie d'abord les grandes tâches distantes, en fonction de ses connaissances et croyances locales. Cette stratégie s'appuie sur une structure de données appelée lot par possession.

Premièrement, le ratio local de possession mesure la localité d'une tâche :

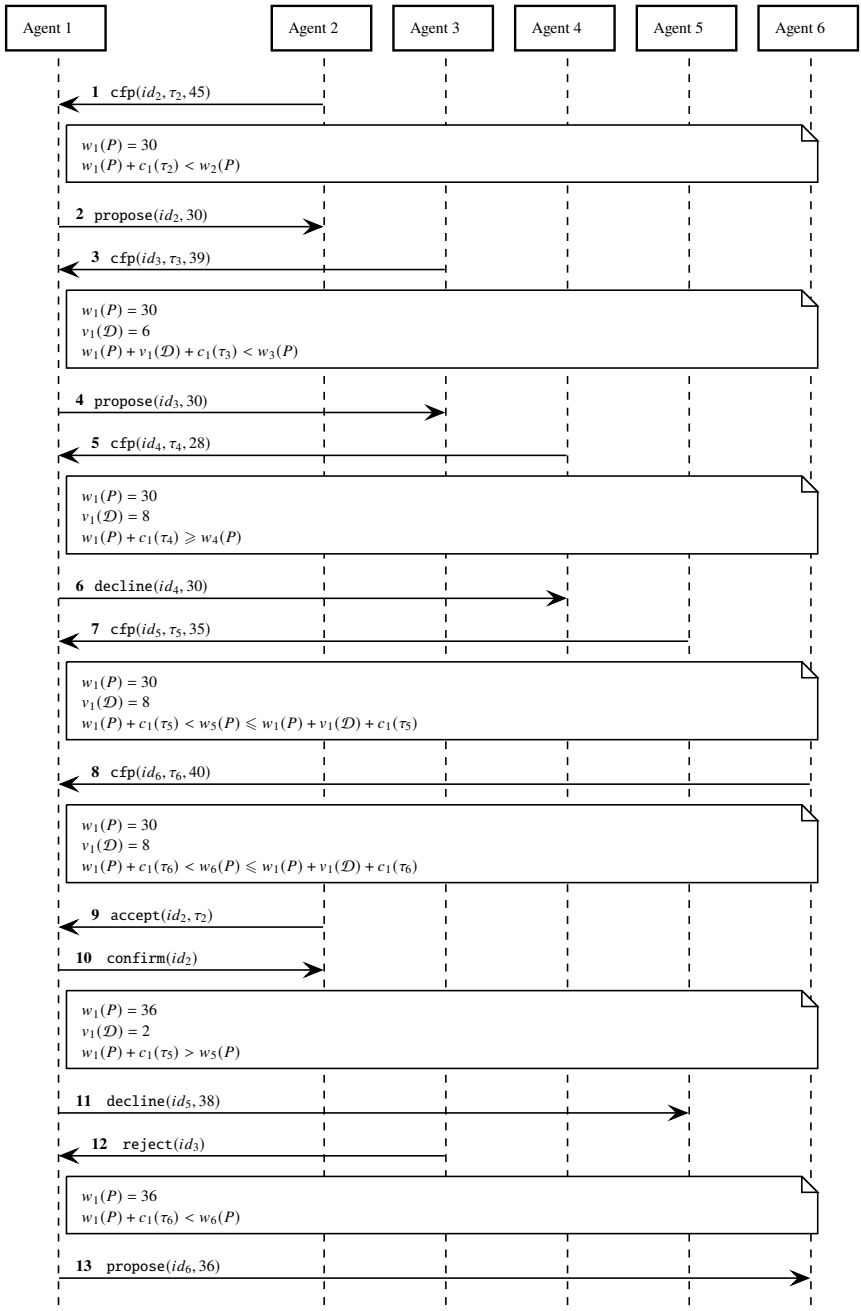


FIGURE 4.2 – L'agent 1 enchérit dans plusieurs délégations concurrentes comme décrit dans l'exemple 4.3.

Définition 5.2 (ratio local de possession). — *Le ratio local de possession d'un agent i pour une tâche τ est défini comme :*

$$o_i(\tau) = \frac{d_i(\tau)}{d_\tau} \tag{5.1}$$

Le ratio maximal de possession pour la tâche τ est :

$$\hat{o}(\tau) = max_{i \in \mathcal{A}}\{o_i(\tau)\} \tag{5.2}$$

Le ratio local de possession d'un agent pour une tâche est le ratio entre le nombre de ressources locales à l'agent pour cette tâche et le nombre total de ressources de la tâche.

Deuxièmement, le **lot par possession** d'un agent i, illustré par la figure 5.1, se décompose en trois paquets de tâches selon le ratio local de possession de l'agent pour ces tâches.

(1) *le paquet local* contient les tâches pour lesquelles l'agent i possède au moins une ressource et pour lesquelles il n'y a pas un autre agent qui possède plus de ressources que lui pour cette tâche. Les tâches sont triées par ordre décroissants de leur coût (voir figure 5.1 à gauche) ;
(2) *le paquet semi-local* contient les tâches qui sont partiellement locales. Les tâches sont triées par ordre décroissant de ratio local de possession et les tâches avec le même ratio sont triées par ordre de coût décroissant (voir figure 5.1 au centre) ;
(3) *Le paquet distant* contient les tâches qui sont distantes. Les tâches sont triées par ordre de coût croissant (voir figure 5.1 à droite).

La stratégie situationnelle sélectionne la tâche à réaliser en considérant d'abord la première tâche du paquet local, qui correspond à la plus grande tâche locale. Pour identifier la tâche à négocier, la stratégie situationnelle commence par la dernière tâche du paquet distant, ce qui correspond à la plus coûteuse des tâches distantes, et sélectionne la première tâche qui permet une délégation potentiellement socialement rationnelle.

L'exemple suivant illustre la différence de comportement de ces deux stratégies.

Exemple 5.3. — Considérons le problème $MASTA^{ex}$ et l'allocation P de l'exemple 3.3. Le lot de l'agent 2 est $\{\tau_1, \tau_3, \tau_5, \tau_7\}$.

Si l'agent 2 adopte la stratégie agnostique, il exécute ses tâches dans l'ordre suivant : τ_1, τ_5, τ_7, puis τ_3. Les délégations sont envisagées dans l'ordre inverse.

Si l'agent 2 adopte la stratégie situationnelle, l'ordre d'exécution est τ_7, τ_5, τ_3 puis τ_1. En effet, la tâche τ_7 est locale alors que la tâche τ_1 est distante. Les tâches τ_5 et τ_3 sont semi-locales avec des ratio locaux de possession égaux à $\frac{4}{5}$ et à $\frac{6}{9}$ respectivement.

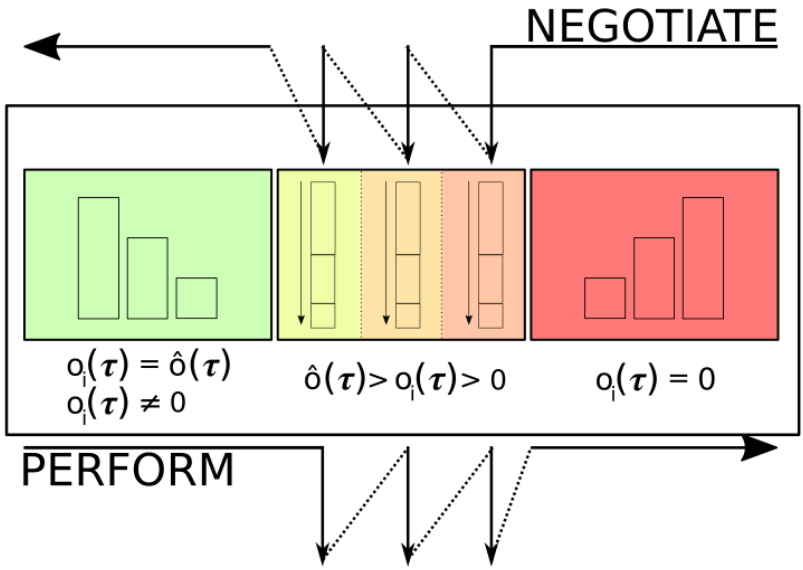


FIGURE 5.1 – Le lot par possession contient le paquet local (à gauche), le paquet semi-local (au centre) et le paquet distant (à droite). Les tâches sont représentées par des rectangles dont la taille est proportionnelle au coût de la tâche. Les flèches désignent l'ordre de parcours selon lequel un agent va chercher une tâche à exécuter/négocier.

6. RÉSULTATS ET DISCUSSION

Après avoir présenté notre application pratique, nous décrivons notre prototype et discutons des résultats expérimentaux.

6.1. APPLICATION PRATIQUE

L'application pratique que nous considérons est le déploiement distribué du patron de conception MapReduce afin de traiter de grands volumes de données sur un cluster [17], comme le fait Hadoop [44]. Un *job* MapReduce consiste en deux phases successives appelées *map* et *reduce*. Pendant la phase de *map*, les nœuds filtrent en parallèle les données fournies en entrée et génèrent des paires clefs-valeurs qu'ils écrivent dans des fichiers appelés *chunks*. Chaque *chunk* est localisé sur le nœud où a été exécutée l'opération *map* qui l'a généré. Pendant la phase de *reduce*, les nœuds traitent en parallèle les clefs et leurs listes de valeurs.

La phase de *reduce* d'un *job* MapReduce peut être formalisée par un problème MASTA $\langle Node, \mathcal{A}, \mathcal{T}, l, d, c\rangle$ (cf. définition 3.1) où : *Node* est l'ensemble des nœuds du cluster , $\mathcal{A}$ est l'ensemble des agents *reducers*, $\mathcal{T}$ est l'ensemble des tâches, l capture le fait que nous plaçons un *reducer* par nœud et d concerne la localisation des données. En conformité avec l'équation 3.1, nous spécifions le coût d'une tâche τ pour

un agent i ainsi :

$$c_i(\tau) = \Sigma_{\rho \text{ est un chunk pour } \tau} c_i(\rho), \text{ avec } c_i(\rho) = \begin{cases} |\rho| & \text{si } \rho \text{ est local pour } i \\ \kappa \times |\rho| & \text{sinon} \end{cases} \quad (6.1)$$

où $|\rho|$ représente le nombre de valeurs pour le *chunk* ρ et κ capture le temps de récupération de la ressource. Expérimentalement nous fixons $\kappa = 2$ pour un cluster et $\kappa = 10$ quand nous utilisons un réseau d'ordinateurs. Nos expériences montrent que la fonction de coût n'a pas besoin d'être adaptée finement puisque l'adaptabilité de notre processus de réallocation dynamique de tâches compense une fonction de coût imprécise.

6.2. Implémentation

Nous avons développé une version multi-agents du patron MapReduce dans un système distribué utilisant le banc d'essai MAS4Data [5]. MAS4Data est implementé avec Akka [33] qui est adapté aux applications orientées messages, fortement concurrentes, distribuées et robustes. Même si la tolérance aux pannes des nœuds est hors de la portée de cette étude, nous supposons que le délai de transmission des messages est arbitraire mais non négligeable et que des messages peuvent être perdus. Pour ces raisons, nous avons inclus des mécanismes d'accusé de réception et de date butoire dans le protocole d'interaction. Afin de réduire la complexité due à la conception d'un agent *reducer*, nous avons adopté une architecture multi-agents modulaire qui permet à la fois l'exécution concurrente de la négociation et de l'exécution des tâches, et la séparation des comportements pour la prise de décision ou les communications.

6.3. Résultats empiriques

Dans cette section, dans un premier temps, nous détaillons la configuration expérimentale qui inclut le choix des métriques, les jeux de données et les *jobs*. Dans un second temps, nous exposons les résultats expérimentaux.

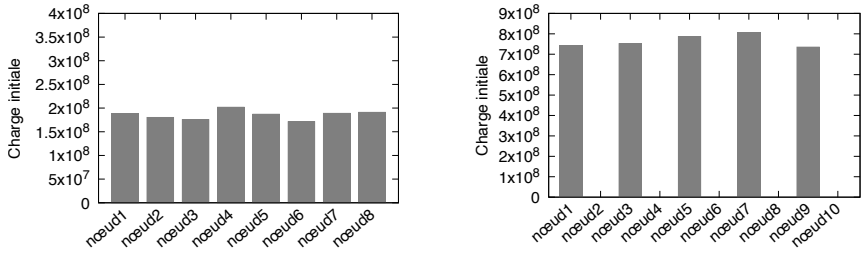
Métriques. — Afin d'évaluer les temps d'exécution et l'équité de nos expériences, les métriques suivantes ont été introduites dans nos précédents travaux [7, 8] :

- la **contribution** d'un *reducer* est la somme des coûts des tâches qu'il a réalisées au cours de l'exécution du *job*. Elle est donc mesurée *ex-post* ;
- l'**équité de contribution** est le rapport entre la plus petite et la plus grande contribution des agents ;
- le **temps d'exécution** de la phase de *reduce* est le temps d'exécution de l'agent qui termine en dernier ;
- l'**équité en temps** est le rapport entre les temps d'exécution de l'agent le plus lent et du plus rapide.

La contribution d'un *reducer* correspond à sa charge de travail (voir définition 3.2) observée *ex-post*. Elle mesure la quantité de travail réalisé par le *reducer* lors de l'exécution du *job*. Plus la contribution d'un *reducer* est importante, plus ce dernier a

contribué à la réalisation du *job*. Une équité de contribution proche de 1 signifie que la charge de travail a été équitablement répartie entre les agents. Le temps d'exécution est une mesure du *makespan*. Le temps d'exécution d'un agent mesure la durée effective de travail de l'agent entre le début du traitement du *job* et le moment où il finit d'exécuter sa dernière tâche. Une équité en temps proche de 1 signifie que chacun des agents est occupé autant que les autres : il n'y a pas d'agent inactif avant la fin du traitement du *job*. Il est important de noter que l'équité en temps et l'équité en contribution sont deux métriques distinctes. Comme nous le verrons, dans des environnements hétérogènes, le processus de négociation permet de garantir une équité en temps (cf. figure 6.5(b)), chaque agent travaillant autant de temps que les autres malgré des capacités hétérogènes qui se traduisent par une équité en contribution faible (cf. figure 6.5(a)) et 6.5(d)).

Configuration. — Nous considérons ici deux configurations matérielles différentes : (a) un cluster de 10 lames, chacune composée de 10 CPUs avec 512Go RAMP ; (b) un réseau de 16 PCs équipés de 4 cœurs Inter(R) i7 et 16 GB de RAM chacun. Dans la mesure où les résultats sont similaires avec la seconde configuration, nous présentons ici les expériences réalisées sur le cluster, à l'exception de la dernière expérience qui porte sur la comparaison entre les deux configurations. Nous utilisons deux jeux de données réels différents. Le premier (2.4 Go) contient 100 480 507 évaluations que 480 189 utilisateurs ont donné à 17 770 films [14]. Le *job*, appelé `RecByMov`, compte le nombre d'évaluations par film. Le second jeu de données (977 Mo) contient 3 963 480 enregistrements météorologiques (identifiant de station, date, température, précipitations, etc.) issus de stations au cours des 20 dernières années [36]. Le *job*, appelé `RecByTempSta`, compte le nombre d'enregistrements par demi-degré de température par station. Afin d'augmenter le volume des données sans allonger la phase de *map* (qui est en dehors du champ de cet article) et sans modifier la distribution des données, nous répliquons k fois les valeurs pour chaque clef, avec $k = 200$ pour le *job* `RecByTempSta` et $k = 20$ pour le *job* `RecByMov`. Quels que soient le *job* et le jeu de données, nous utilisons autant de *mapper* que de *reducers* et chaque expérience est exécutée 30 fois. En raison du non-déterminisme observable des exécutions distribuées, nous commentons des exécutions particulières.



(a) Le *job* `RecByTempSta` sur 8 nœuds

(b) Le *job* `RecByMov` sur 10 nœuds

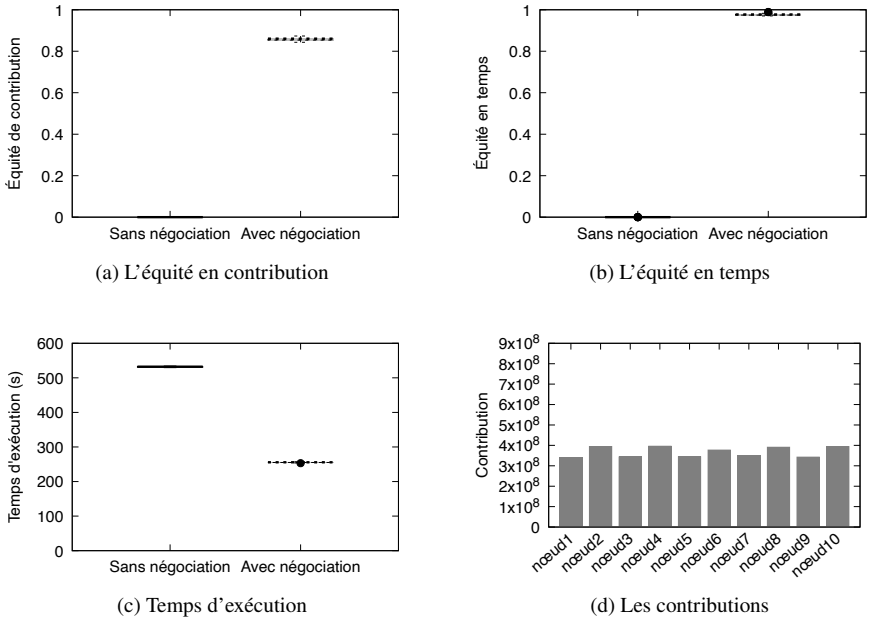
Figure 6.1 – L'allocation de nos deux *jobs* avec le partitionnement classique d'Hadoop

Résultats empiriques. — Dans un premier temps, nos expériences visent à valider notre approche. Nous montrons que : (a) la négociation améliore le *makespan*, et donc le temps d'exécution avec un faible surcoût dû à la négociation, et (b) notre système multi-agents est adaptatif aux variations de performance et aux environnements hétérogènes. Nous utilisons la stratégie agnostique vis-à-vis de la localité et le processus multi-enchères. Dans un second temps, nous évaluons ces choix.

La négociation améliore le temps d'exécution. — La négociation est bénéfique, que l'allocation initiale soit équitable ou non. Afin de valider cette hypothèse, nous comparons l'équité en contribution, l'équité en temps et le temps d'exécution de *jobs* avec et sans négociation. Précisons que, dans toute la suite de l'article, « sans négociation » correspond à « avec la méthode classique d'Hadoop ».

Résultat expérimental 1. — *La négociation améliore le temps d'exécution grâce à l'équilibrage des contributions des reducers.*

La figure 6.2 montre qu'un *job* peut fortement bénéficier de la négociation. En partant d'une allocation inéquitable (cf. figure 6.1b), nous exécutons 30 fois le *job* `RecByMov` avec 10 *reducers* sur 10 nœuds homogènes. Au cours du traitement des données, des délégations de tâche se produisent puisque les charges sont déséquilibrées.



(a) L'équité en contribution

(b) L'équité en temps

(c) Temps d'exécution

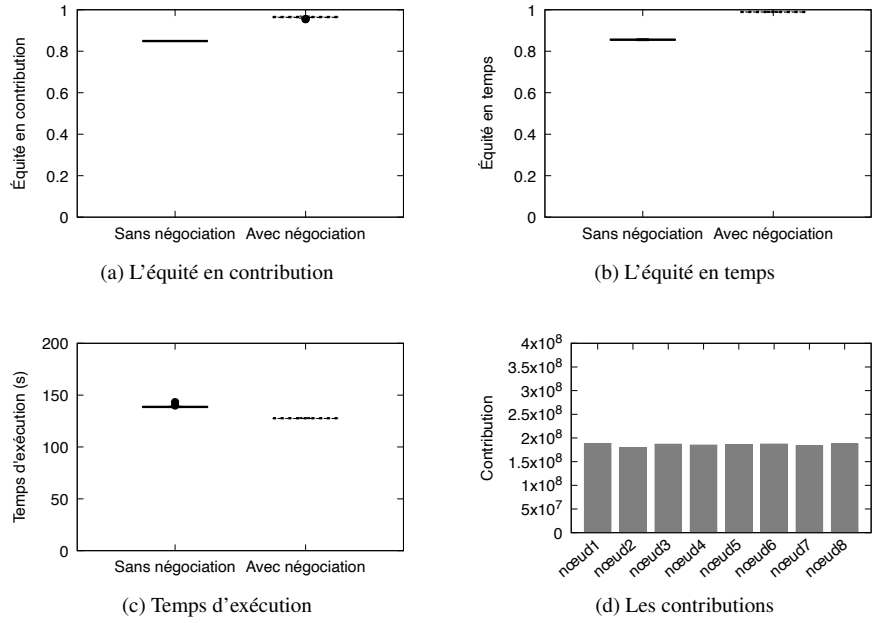
(d) Les contributions

Figure 6.2 – Les métriques médianes et les écarts types représentés par des diagrammes en boîte (a,b,c), et les contributions après négociation dans une exécution du *job* `RecByMov` (d).

Les *reducers* les plus chargés proposent des tâches qui sont acceptées par les moins chargés. À terme, les contributions sont équitablement distribuées entre tous les nœuds et tous les *reducers* terminent au même moment puisque l'équité en temps est proche de 1 (cf. figures 6.2a et 6.2b). En conséquence, le *makespan* (cf. figure 6.2d) et donc le temps d'exécution (cf. figure 6.2c) sont plus que divisés par deux.

RÉSULTAT EXPÉRIMENTAL 2. — *Quand l'allocation initiale est équitable, le surcoût dû à la négociation est négligeable et n'affecte pas le temps d'exécution.*

La figure 6.3 montre qu'un *job* ne peut pas être pénalisé par les négociations. Partant d'une allocation initialement équitable (cf. figure 6.1a), nous exécutons 30 fois le *job* `RecByTempSta` avec 8 *reducers* sur 8 nœuds homogènes. Dans la mesure où l'allocation initiale est équitable, sans surprise l'équité en contribution (cf. figure 6.3a) et l'équité en temps (cf. figure 6.3b) sont plutôt bons même sans négociation, c'est-à-dire proches de 0.85. Cependant, ils sont encore améliorés par la négociation puisque le *makespan* est légèrement diminué (cf. figure 6.3d). En conséquence, le temps d'exécution est plus rapide d'environ 7 % avec la négociation (cf. figure 6.3c). Il est utile de noter qu'aucune négociation n'est déclenchée quand un agent pense que l'allocation est stable.



(a) L'équité en contribution

(b) L'équité en temps

(c) Temps d'exécution

(d) Les contributions

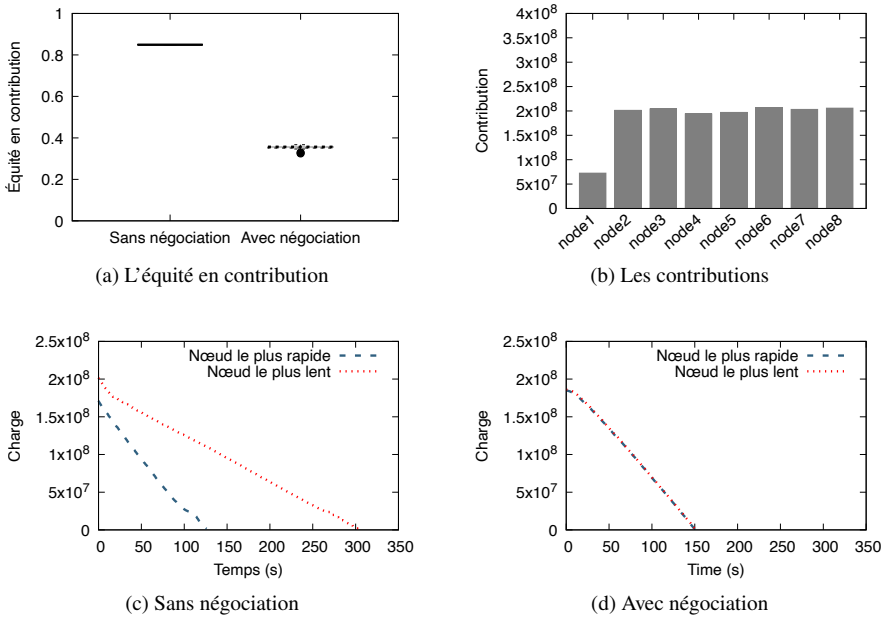
FIGURE 6.3 – Les métriques médianes et leurs écarts types représentés par des diagrammes en boîte (a,b,c), et les contributions après négociation dans une exécution du *job* `RecByTempSta` (d).

Ces deux expériences valident que notre processus adaptatif dynamique basé sur les négociations concurrentes pour la réallocation de tâches améliore le temps d'exécution même lorsque l'allocation initiale est équilibrée. La première expérience démontre que MAS4Data traite le biais de partitionnement. La seconde met en évidence le caractère négligeable du coût de la négociation au regard du bénéfice de l'équilibrage des charges de travail.

La négociation compense l'hétérogénéité. — L'hétérogénéité d'un environnement de calcul peut être due soit à une non uniformité des capacités intrinsèques des nœuds de calcul, soit à des raisons exogènes, telle que la baisse de performance d'un nœud.

Résultat expérimental 3. — *Dans un environnement hétérogène, la négociation permet de réallouer les tâches aux nœuds les plus rapides afin d'améliorer le temps d'exécution.*

La figure 6.4 montre que le système multi-agents adapte l'allocation face aux aléas d'exécution. Partant d'une allocation initialement équitable (cf. figure 6.1a), nous exécutons 30 fois le *job* `RecByTempSta` avec 8 *reducers*. Un seul CPU est activé sur



(a) L'équité en contribution

(b) Les contributions

(c) Sans négociation

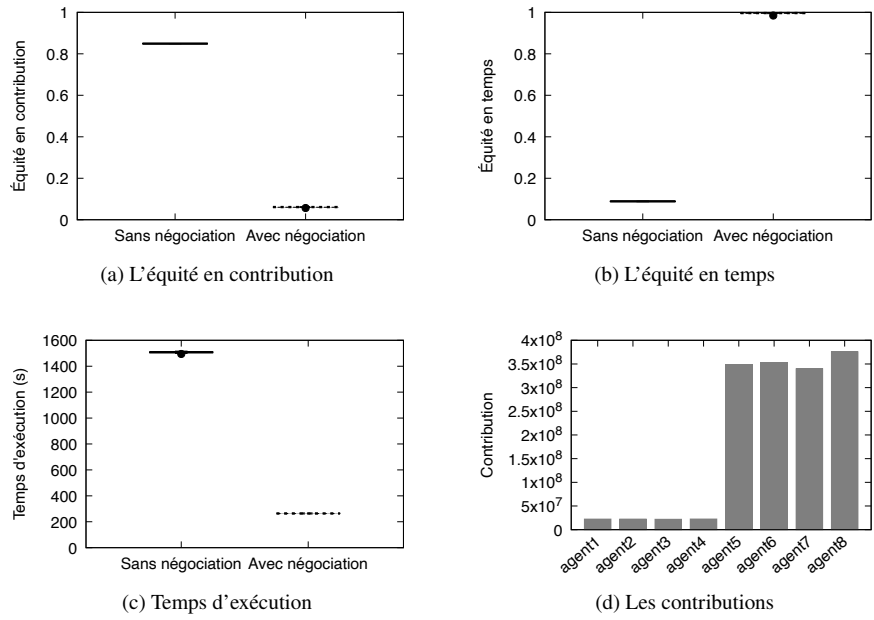
(d) Avec négociation

Figure 6.4 – L'équité en contribution médiane et son écart type sous forme de diagramme de boîte (a), les contributions (b) pour une exécution particulière du *job* `RecByTempSta` quand un nœud est ralenti, l'évolution des charges de travail sans (c) et avec (d) négociations.

chaque nœud et le premier *reducer* ne peut pas utiliser plus de 50 % du temps CPU. Les sept autres *reducers* s'exécutent donc plus rapidement. L'allocation initiale est équilibrée. Elle est remise en cause par le *reducer* le plus lent. Sans négociation (cf. figure 6.4c), sept *reducers* terminent après 130 secondes tandis que le plus lent termine après 300 secondes. Il en résulte que l'équité en temps est faible (environ 0.43). Par contraste, la négociation permet au *reducer* le plus lent de terminer après 150 secondes, en même temps que les autres (cf. figure 6.4d). L'équité en temps est alors très proche de 1 puisque des négociations ont été déclenchées pendant le traitement du *job* pour permettre l'équilibrage. Grâce à ce processus dynamique et continu de réallocation de tâches, le *job* n'est pas pénalisé par le nœud le plus lent et le *job* s'exécute deux fois plus rapidement grâce à la négociation. Enfin, la contribution du *reducer* le plus lent est plus faible que celles des autres (cf. figure 6.4b), l'équité en contribution est donc faible, approximativement 0.35 (cf. figure 6.4a).

Cette expérience montre que le processus de négociation atténue l'impact d'un aléa d'exécution qui ralentirait un *reducer*. Un *reducer* ralenti peut déléguer certaines tâches afin que le *job* puisse être exécuté aussi vite que possible.

La figure 6.5 montre que le système multi-agents adapte l'allocation à un environnement hétérogène. Partant d'une allocation initialement équitable (cf. figure 6.1a),



(a) L'équité en contribution

(b) L'équité en temps

(c) Temps d'exécution

(d) Les contributions

FIGURE 6.5 – Les métriques médianes et écarts types représentés par des diagrammes en boîtes (a,b,c), et les contributions après négociation dans une exécution du *job* `RecByTempSta` dans un environnement hétérogène (d).

nous exécutons 30 fois le *job* `RecByTempSta` avec 8 *reducers*. Un seul CPU est activé sur chaque nœud. Quatre *reducers* s'exécutent sur le même nœud et quatre sur des nœuds séparés. Comme précédemment, l'équité en contribution est détériorée puisque les quatre *reducers* les plus lents exécutent moins de tâches que les quatre autres. Une fois encore, le processus dynamique et continu de réallocation de tâches permet d'atteindre une équité en temps proche de 1 et le *job* s'exécute cinq fois plus vite grâce aux négociations. Enfin, les contributions des *reducers* les plus lents sont plus faibles que celles des autres, l'équité en contribution est faible.

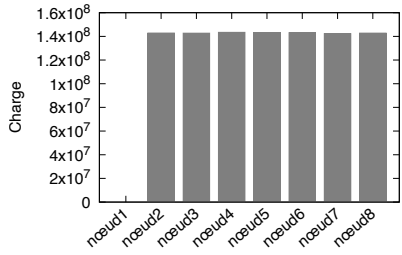
Ces deux expériences valident qu'un *job* qui s'exécute dans un environnement hétérogène bénéficie de l'adaptativité de notre système multi-agents. Quand les variations de performance provoquent une allocation déséquilibrée, notre processus dynamique et continu le détecte et déclenche la réallocation de tâches en faveur des nœuds les plus rapides entraînant une réduction du temps d'exécution.

Enchères uniques contre multi-enchères. — Nous supposons que notre processus multi-enchères, qui permet à un agent d'enchérir dans plusieurs négociations concurrentes, améliore la réactivité de notre système multi-agents.

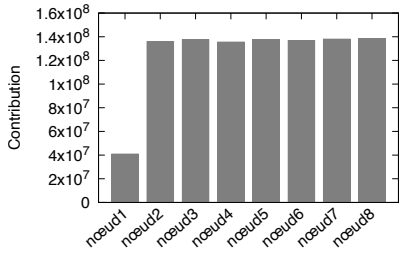
Résultat expérimental 4. — *Le processus multi-enchères permet d'atteindre plus rapidement une allocation stable.*

La figure 6.6 montre que le système multi-agents adapte plus rapidement l'allocation avec le processus multi-enchères. Nous avons généré un jeu de données (775 Mo) avec 100 000 000 lignes de sorte que l'allocation initiale soit inéquitable. Le *job*, appelé `RecByKey`, compte le nombre de valeurs par clef. Nous utilisons 8 *reducers* dans un environnement homogène où aucune tâche n'est attribuée au premier *reducer* alors que 100 000 tâches avec 1 000 valeurs par tâche sont attribuées aux autres *reducers* qui ont donc des charges de travail similaires (cf. figure 6.6a). Comme dans les expériences précédentes, la négociation améliore l'équilibrage des charges (cf. figure 6.6b et 6.6c). Cependant, le processus multi-enchères est plus efficace. Alors que ce processus nécessite 50 secondes pour atteindre une allocation équitable (cf. figure 6.6e), la charge de travail du premier agent reste proche de 0 pendant toute la phase de *reduce* pour le premier agent avec le processus d'enchère unique (cf. figure 6.6d). En fait, dans le processus d'enchère unique, les tâches sont déléguées au premier *reducer* les unes après les autres et elles sont immédiatement consommées. Par contraste, dans le cas du processus multi-enchères, cet agent enchérit en continu simultanément dans sept enchères. En conséquence, le traitement des données avec les enchères uniques ($\sim$ 275s) est environ 10 % plus lent qu'avec les multi-enchères ($\sim$ 245s). Enfin, l'équité en contribution est d'environ 0.3 avec les enchères uniques alors qu'elle est proche de 0.85 avec le processus multi-enchères.

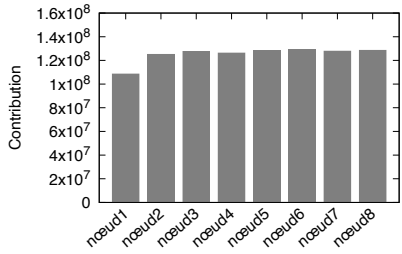
Cette expérience montre que le processus multi-enchères accélère le processus d'équilibrage des charges de travail, améliorant ainsi la réactivité du système multi-agents.



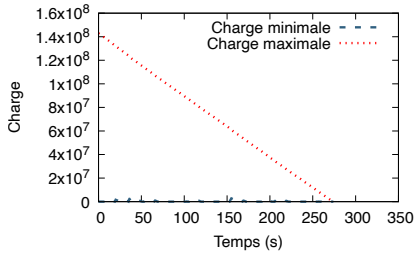
(a) Les charges initiales



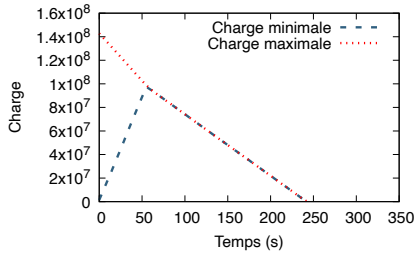
(b) Les contributions avec des enchères uniques



(c) Les contributions avec des multi-enchères



(d) Les charges avec des enchères uniques



(e) Les charges avec des multi-enchères

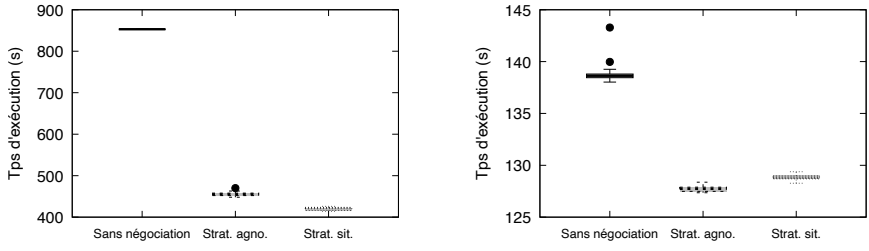
FIGURE 6.6 – Les charges de travail initiales (a), les contributions et les charges de travail pour une exécution particulière du job *RecByKey* dans le cas d'enchères uniques (b et d) et de multi-enchères (c et e).

ADÉQUATION DE LA STRATÉGIE AVEC L'INFRASTRUCTURE. — Nous supposons que l'efficacité de la stratégie situationnelle dépend du temps de récupération des ressources.

RÉSULTAT EXPÉRIMENTAL 5. — *La stratégie situationnelle améliore le temps d'exécution quand le coût de récupération des ressources est significatif.*

La figure 6.7 montre que l'efficacité de la stratégie de négociation dépend de la configuration matérielle. Nous avons généré un jeu de données (8 Go) avec 82 283 clefs

tel que l'allocation initiale des tâches du *job* `RecByKey` puisse être remise en cause. Afin d'évaluer l'impact sur l'exécution de la proximité entre les ressources et les nœuds de calcul, la plupart des données nécessaires à une tâche ne sont pas localisées sur le même nœud que le *reducer* (voir [8] pour plus de détails). Dans un réseau d'ordinateurs, la stratégie situationnelle améliore d'environ −7.6 % le temps d'exécution par rapport à la stratégie agnostique vis-à-vis de la localité (cf. figure 6.7a). Par contraste, la stratégie agnostique vis-à-vis de la localité est plus efficace avec un cluster (cf. figure 6.7b). En effet, le coût de récupération des ressources distantes auprès des autres nœuds dans un réseau d'ordinateurs a un réel impact. Il en résulte que l'exécution de tâches locales améliore l'exécution. À l'inverse, ce surcoût d'exécution de la tâche est faible au sein d'un cluster. Nous le mesurons expérimentalement aux alentours de 10 %. De plus, il est utile de noter que la localité est prise en compte implicitement par la stratégie agnostique vis-à-vis de la localité puisque la fonction de coût est définie de telle sorte qu'une tâche soit moins coûteuse quand les ressources nécessaires sont locales. (cf. Équation 3.1).



(a) Réseau d'ordinateurs (b) Cluster de calcul

FIGURE 6.7 – Les temps d'exécution pour un *job* donné sur un réseau d'ordinateurs (a) et un cluster (b).

7. CONCLUSION

Dans cet article nous avons proposé un système multi-agents pour la réallocation de tâches entre des nœuds distribués, basée sur la localisation des ressources nécessaires à la réalisation de ces tâches, afin de minimiser le *makespan*. En particulier, nous avons appliqué notre système de négociation pour équilibrer la phase de *reduce* du modèle distribué MapReduce pour pouvoir traiter de grands jeux de données.

Notre prototype a été évalué expérimentalement. Nos expériences montrent que MAS4Data est adaptatif au biais de partitionnement, à un environnement de calcul hétérogène et aux aléas d'exécution. Cela est rendu possible par la négociation qui améliore le temps d'exécution en équilibrant les contributions des *reducers*. Certaines de ces expériences suggèrent que les travaux futurs doivent prendre en compte : (a) l'échange de tâches pour améliorer le *makespan* d'allocations stables et (b) la négociation de paquets de tâches pour accélérer le processus de négociation. MAS4Data passe

à l'échelle puisqu'il permet de traiter un grand nombre de tâches grâce à des décisions locales des agents sur les prochaines tâches à réaliser/déléguer. De plus, le surcoût de la négociation est négligeable par rapport au bénéfice de l'équilibre de charge puisque la réallocation de tâches est concurrente avec la consommation de tâches et qu'aucune négociation n'est déclenchée quand les agents estiment que l'allocation est stable. En effet, notre méthode n'est pas un algorithme d'ordonnancement qui alloue les tâches une fois pour toute mais une stratégie distribuée qui cherche en continu à réparer une partition potentiellement déséquilibrée.

Du point de vue de l'utilisateur, même si notre approche adaptative et dynamique s'attaque au problème de la baisse de performance, la tolérance aux pannes peut encore être obtenue par la réplication des données. Il est important de noter que, grâce à son adaptabilité, MAS4Data ne nécessite pas de paramétrage issue d'une expertise. Même si le choix de la stratégie dépend de la configuration matérielle, aucun autre paramètre ne nécessite d'être finement adapté, comme le facteur de réplication (par défaut 3 dans HDFS [44]). Une analyse de sensibilité de ce paramètre est hors du champ de ce travail, mais nécessiterait certainement d'être explorée.

Plus généralement, ce travail doit se poursuivre en considérant l'arrivée en continu de *jobs* concurrents soumis par différents utilisateurs alors que notre étude actuelle se focalise sur la réaffectation de tâches indépendantes dans un *job* unique, pendant leur exécution. Pour combler cet écart de granularité, notre cadre formel doit être étendu et le critère d'optimisation à prendre en compte devrait être le temps de réalisation de plusieurs *jobs* concurrents.

8. Remerciements

Nous remercions le comité de programme des journées francophones sur les systèmes multi-agents ainsi que le comité éditorial de la revue ouverte en intelligence artificielle qui, par leurs remarques, nous ont permis d'améliorer cet article.

Bibliographie

[1] A. Agnetis, J. Billaut, S. Gawiejnowicz, D. Pacciarelli & A. Soukhal, *Multiagent Scheduling – Models and Algorithms*, Springer, 2014.

[2] B. Alrayes, Ö. Kafali & K. Stathis, « Concurrent bilateral negotiation for open e-markets: the CONAN strategy », *Knowledge and Information Systems* **56** (2017), nº 2, p. 463-501.

[3] B. An, V. Lesser, D. Irwin & M. 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)*, 2010, p. 981-988.

[4] G. Attiya & Y. Hamam, « Task allocation for maximizing reliability of distributed systems: A simulated annealing approach », *Journal of Parallel and Distributed Computing* **66** (2006), nº 10, p. 1259-1266.

[5] Q. Baert, A.-C. Caron, M. Morge & J.-C. Routier, « MAS4Data: Multiagent systems for processing very large datasets », `https://github.com/cristal-smac/mas4data`, visited 2019-12-17.

[6] Q. Baert, A.-C. Caron, M. Morge & J.-C. Routier, « Stratégie de découpe de tâche pour le traitement de données massives », in *Actes des 25èmes journées francophones sur les systèmes multi-agents*, Cépaduès, 2017, p. 65-74.

[7] Q. Baert, A.-C. Caron, M. Morge & J.-C. Routier, « Fair multi-agent task allocation for large datasets analysis », *Knowledge and Information Systems* **54** (2018), nº 3, p. 591-615.

[8] Q. Baert, A.-C. Caron, M. Morge, J.-C. Routier & K. Stathis, « A Location-Aware Strategy for Agents Negotiating Load-balancing », in *Proc. of the 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019.

[9] Q. Baert, A.-C. Caron, M. Morge, J.-C. Routier & K. Stathis, « Stratégie situationnelle pour l'équilibrage de charge », in *Actes des 27ièmes journées francophones sur les systèmes multi-agents*, Cépaduès, 2019, p. 9-18.

[10] 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.

[11] S. Banerjee & J. P. Hecker, « A Multi-agent System Approach to Load-Balancing and Resource Allocation for Distributed Computing », in *First Complex Systems Digital Campus World E-Conference 2015*, Springer International Publishing, 2017, p. 41-54.

[12] 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)* (Bordeaux, France), Cépaudès, 2021, p. 1-10.

[13] ———, « 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)*, JFSMA 2021. Collectifs cyber-physiques, Cépaduès, 2021, p. 51-60.

[14] J. Bennett, S. Lanning et al., « The netflix prize », in *Proceedings of KDD cup and workshop*, vol. 2007, 2007, p. 35.

[15] B. Chen, C. N. Potts & G. J. Woeginger, « Handbook of combinatorial optimization », chap. A review of machine scheduling: Complexity, algorithms and approximability, p. 1493-1641, Springer, 1998.

[16] Q. Chen, D. Zhang, M. Guo, Q. Deng & S. Guo, « SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment », in *Proc. of the 14th International Conference on Computer and Information Technology*, 2010, p. 2736-2743.

[17] J. Dean & S. Ghemawat, « MapReduce: Simplified Data Processing on Large Clusters », in *Proc. of the 9th Symposium on Operating Systems Design and Implementation*, 2004, p. 137-150.

[18] U. Endriss, N. Maudet, F. Sadri & F. Toni, « Negotiating Socially Optimal Allocations of Resources », *Journal of Artificial Intelligence Research* **25** (2006), nº 1, p. 315-348.

[19] S. K. Garg, S. Venugopal, J. Broberg & R. Buyya, « Double auction-inspired meta-scheduling of parallel applications on global grids », *Journal of Parallel and Distributed Computing* **73** (2013), nº 4, p. 450-464.

[20] M. Hammoud, M. Rehman & M. Sakr, « Center-of-Gravity Reduce Task Scheduling to Lower MapReduce Network Traffic », in *Proc. of the 5th International Conference on Cloud Computing (CLOUD)*, 2012, p. 49-58.

[21] A. M. A. Hariri & N. Potts, Chris, « Heuristics for scheduling unrelated parallel machines », *Computers & operations research* **18** (1991), nº 3, p. 323-331.

[22] E. Horowitz & S. Sahni, « Exact and approximate algorithms for scheduling nonidentical processors », *Journal of the ACM* **23** (1976), nº 2, p. 317-327.

[23] O. H. Ibarra & C. E. Kim, « Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors », *Journal of ACM* **24** (1977), nº 2, p. 280-289.

[24] 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.

[25] Y. Jiang & Z. Huang, « The rich get richer: Preferential attachment in the task allocation of cooperative networked multiagent systems with resource caching », *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* **42** (2012), nº 5, p. 1040-1052.

[26] Y. Jiang & J. Jiang, « Contextual resource negotiation-based task allocation and load balancing in complex software systems », *IEEE Transactions on Parallel and Distributed Systems* **20** (2008), nº 5, p. 641-653.

[27] Y. Jiang & Z. Li, « Locality-sensitive task allocation and load balancing in networked multiagent systems : Talent versus centrality », *Journal of Parallel and Distributed Computing* **71** (2011), nº 6, p. 822-836.

[28] Y. Jiang, Y. Zhou & W. Wang, « Task allocation for undependable multiagent systems in social networks », *IEEE Transactions on Parallel and Distributed Systems* **24** (2012), nº 8, p. 1671-1681.

[29] S. Kraus, O. Shehory & G. Taase, « Coalition formation with uncertain heterogeneous information », in *Proc. of 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2003, p. 1-8.

[30] Y. Kwon, M. Balazinska, B. Howe & J. Rolia, « Skewtune: mitigating skew in mapreduce applications », in *Proc. of the SIGMOD International Conference on Management of Data*, ACM, 2012, p. 25-36.

[31] Y. Kwon, K. Ren, M. Balazinska & B. Howe, « Managing Skew in Hadoop », *IEEE Data Eng. Bull.* **36** (2013), nº 1, p. 24-33.

[32] J. K. Lenstra, D. B. Shmoys & E. Tardos, « Approximation algorithms for scheduling unrelated parallel machines », *Mathematical programming* **46** (1990), nº 1-3, p. 259-271.

[33] Lightbend, Inc, « Akka toolkit », `https://akka.io`, visited 2019-12-17.

[34] M. Liroz-Gistau, R. Akbarinia & P. Valduriez, « FP-Hadoop: efficient execution of parallel jobs over skewed data », *VLDB Endowment* **8** (2015), nº 12, p. 1856-1859.

[35] S. Martello, F. Soumis & P. Toth, « Exact and approximation algorithms for makespan minimization on unrelated parallel machines », *Discrete applied mathematics* **75** (1997), nº 2, p. 169-188.

[36] Météo France, « Données SYNOP essentielles OMM », 2019, `https://donneespubliques.meteofrance.fr/?fond=produit&id_produit=90&id_rubrique=32`.

[37] E. Mokotoff, « Parallel machine scheduling problems: A survey », *Asia-Pacific Journal of Operational Research* **18** (2001), nº 2, p. 193-242.

[38] S. Penmatsa & A. T. Chronopoulos, « Game-theoretic static load balancing for distributed systems », *Journal of Parallel and Distributed Computing* **71** (2011), nº 4, p. 537-555.

[39] 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.

[40] M. Schillo, C. Kray & K. Fischer, « The eager bidder problem: a fundamental problem of DAI and selected solutions », in *Proc. of 1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2002, p. 599-606.

[41] O. Selvitopi, G. V. Demirci, A. Turk & C. Aykanat, « Locality-aware and load-balanced static task scheduling for MapReduce », *Future Generation Computer Systems* **90** (2019), p. 49-61.

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

[43] R. G. Smith, « The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver », *IEEE Transactions on computers* **29** (1980), nº 12, p. 1104-1113.

[44] The Apache Software Foundation, « Apache Hadoop », `https://hadoop.apache.org`, visited 2019-07-01.

[45] J. Turner, Q. Meng, G. Schaefer & A. 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)*, 2018, p. 739-747.

[46] W. E. Walsh & M. P. Wellman, « A market protocol for decentralized task allocation », in *Proc. of the International Conference on Multi-Agent Systems (ICMAS)*, 1998, p. 325-332.

ABSTRACT. — We study the problem of task reallocation for load-balancing of MapReduce jobs in applications that process large datasets. In this context, we propose a novel strategy based on cooperative agents used to optimise the task scheduling in a single MapReduce job. The novelty of our strategy lies in the ability of agents to identify opportunities within a current unbalanced allocation, which in turn trigger concurrent and one-to-many negotiations amongst agents to locally reallocate some of the tasks within a job. Our contribution is that tasks are reallocated according to the proximity of the resources and they are performed in accordance to the capabilities of the nodes in which agents are situated. To evaluate the adaptivity and responsiveness of our approach, we implement a prototype test-bed and conduct a vast panel of experiments in a heterogeneous environment and by exploring varying hardware configurations. This extensive experimentation reveals that our strategy significantly improves the overall runtime over the classical Hadoop data processing.

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

Manuscrit reçu le 11 mai 2021, révisé le 31 janvier 2022, accepté le 7 février 2022.