Stratégie situationnelle pour l'équilibrage de charge

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

Stratégie situationnelle pour l'équilibrage de charge

Quentin Baert[a] quentin.baert@univ-lille.fr
Anne-Cécile Caron[a] anne-cecile.caron@univ-lille.fr
Maxime Morge[a] maxime.morge@univ-lille.fr
Jean-Christophe Routier[a] jean-christophe.routier@univ-lille.fr
Kostas Stathis[b] kostas.stathis@rhul.ac.uk

[a]Centre de Recherche en Informatique, Signal et Automatique de Lille,
Université de Lille, France

[b]Department of Computer Science,
Royal Holloway, University of London, Egham, United Kingdom

Résumé

Nous étudions une stratégie qui tient compte de la localité des ressources pour équilibrer les charges dans un système distribué. Cette stratégie permet aux agents coopératifs d'identifier une allocation non équilibrée, voire de déclencher des enchères concurrentes pour réallouer localement certaines des tâches. Les tâches sont réallouées en tenant compte de l'accessibilité des ressources pour les agents ; elles sont exécutées conformément aux capacités des nœuds de calcul sur lesquels se trouvent les agents. Ce processus de négociation dynamique et continu est concurrent à l'exécution des tâches, ce qui permet d'adapter l'allocation des tâches aux perturbations (exécution de tâche, chute de performance d'un nœud). Nous évaluons cette stratégie dans le cadre du déploiement multi-agents de MapReduce. Ce patron de conception permet le traitement distribué de données massives. Les résultats empiriques démontrent que notre stratégie améliore significativement le temps d'exécution du traitement d'un jeu de données.

Mots-clés : *Résolution collective de problème, Négociation, Modèles de comportement agent*

Abstract

We study a novel location-aware strategy for distributed systems where cooperating agents perform the load-balancing. The strategy allows agents to identify opportunities within a current unbalanced allocation, which in turn triggers concurrent and one-to-many negotiations amongst agents to locally reallocate some tasks. The tasks are reallocated according to the proximity of the resources and they are performed in accordance with the capabilities of the nodes in which agents are situated. This dynamic and ongoing negotiation process takes place concurrently with the task execution and so the task allocation process is adaptive to disruptions (task consumption, slowing down nodes). We evaluate the strategy in a multi-agent deployment of the MapReduce design pattern for processing large datasets. Empirical results demonstrate that our strategy significantly improves the overall runtime of the data processing.

Keywords: *Distributed cooperative problem solving, Negotiation, Agent behaviour*

1 Introduction

Le problème d'équilibrage des charges et d'allocation des tâches dans un système distribué apparait dans de nombreuses applications telles que l'informatique en nuage, les réseaux pair-à-pair, les réseaux sociaux et le traitement de données massives. Dans cet article, nous abordons les problèmes applicatifs où (a) certaines ressources, par exemple des données, nécessaires à l'exécution d'une tâche sont distribuées sur différents nœuds du réseau, et (b) certains des nœuds sont susceptibles de rencontrer des perturbations lors de l'exécution, par exemple une chute de performance ou des latences réseau. Comme plusieurs ressources sont requises pour exécuter une tâche, toute allocation implique inévitablement le transfert de ces ressources entre les nœuds de calculs, ce qui induit un délai supplémentaire lors du traitement de la tâche [15]. Dans cette classe de problèmes, l'allocation de tâches peut être remise en cause lors de l'exécution des tâches et tirer parti de la distribution des ressources dans le système.

Afin d'aborder le problème d'équilibrage des charges 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, particulièrement celles qui portent sur l'ordonnancement à l'aide d'enchères [27, 24, 26]. En plus de la décentralisation qui permet d'éviter les goulots d'étranglement en terme de performance, nous montrons ici qu'une approche multi-agents pour l'allocation située de tâches répond à deux autres exigences cruciales : (a) la co-occurence de la ré-

allocation des tâches et de leur exécution ; (b) l'adaptation de l'allocation, c'est-à-dire le déclenchement d'une réallocation quand une perturbation se produit. Nous supposons que les agents sont coopératifs. Ils partagent le même objectif : diminuer le temps d'exécution global des tâches. Nous supposons également que les agents ne partagent aucune connaissance, y compris l'état de l'allocation courante. Néanmoins, les agents possèdent un modèle de leurs pairs : ils sont ainsi capables de calculer le coût des tâches pour leurs pairs. Nous supposons également qu'une tâche peut être exécutée par n'importe quel agent seul, et ce sans préemption ni contrainte de précédence. Une tâche est indivisible, sans date butoir et non partageable, c'est-à-dire qu'une tâche n'appartient qu'à un agent à la fois.

Nous formalisons ici le problème d'allocation de tâches situées. En adoptant une approche fondée sur le marché, nous décentralisons totalement le processus d'équilibre des charges grâce auquel les agents coopératifs essaient de minimiser le temps d'exécution de la dernière tâche, i.e. le *makespan*. À cette fin, nous proposons une stratégie situationnelle afin de procéder à l'équilibrage des charges [1]. Quand les agents identifient des opportunités pour rééquilibrer l'allocation courante, ils initient des enchères de manière concurrente pour réallouer localement certaines tâches. Ces dernières sont réallouées en considérant la proximité entre les ressources nécessaires à leur exécution et le nœud sur lequel elles sont effectivement exécutées. Ce processus de réallocation de tâches est dynamique et continu. Il est concurrent à l'exécution des tâches, ce qui rend le système distribué adaptatif aux événements perturbateurs (consommation de tâche, chute de performance d'un nœud).

Notre application pratique est la distribution de MapReduce. Ce patron de conception, dont Hadoop est une implémentation, permet de traiter de larges volumes de données sur des grappes de calculs [9]. Plusieurs biais apparaissent sur des jeux de données réels et mènent à un déséquilibre des charges [17]. Un ordonnanceur centralisé ne peut pas être utilisé comme point de référence de par le grand nombre de tâches à traiter. Cependant, un processus de négociation multi-agents permet de réallouer les tâches lors de la phase de *reduce* pour réduire le temps d'exécution du *job*. Nos résultats empiriques préliminaires montrent que la stratégie situationnelle permet d'améliorer significativement le temps de traitement d'un jeu de données.

Cet article est structuré comme suit. Après une présentation des travaux connexes en section 2, nous formalisons le problème d'allocation de tâches situées à l'aide d'un système multi-agents en section 3. Cette section définit également 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 itérées qui a lieu en même temps que l'exécution des tâches. La section 5 décrit la stratégie situationnelle, 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

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 [6]. 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 [13]. Les algorithmes pseudo-polynomiaux conçus pour ce problème incluent : l'heuristique *earliest completion time* [14] (ECT), les heuristiques de recherche locale [12], l'algorithme *branch and bound* [20] et les heuristiques biphasées basées sur la programmation linéaire [18]. Même si ECT est un algorithme d'approximation qui atteint des résultats acceptables avec un coût computationnel faible, les algorithmes centralisés ne peuvent pas être appliqués à notre scénario où les tâches sont nombreuses (par exemple $82,283$ clés dans la section 6). La stratégie situationnelle est une heuristique de recherche locale décentralisée.

L'ordonnancement multi-agents [15] 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 de par ses exigences :

— **passage à l'échelle.** Un contrôleur global constitue un goulot d'étranglement en terme de performance car il doit collecter les informations d'état de l'ensemble du système en temps réel. Au lieu de

1. Cet article est une version révisée, étendue et traduite de [5] où, dans la continuité de [4], la localité des ressources est prise en compte.

cela, la répartition des tâches peut être négociée par des agents représentants les nœuds [27, 1] ;

— **réactivité.** Les problèmes d'ordonnancement classiques sont statiques. L'estimation inexacte du temps d'exécution des tâches, aggravée par des perturbations (consommation de tâches, ralentissement des nœuds, etc.), peuvent nécessiter d'importantes modifications de l'allocation existante pour qu'elle reste optimale [26]. Plutôt que de recalculer en continu une allocation optimale, quelques modifications locales au cours de l'exécution des tâches peuvent améliorer l'équilibre de charge [3, 28, 16].

La plupart des travaux existants adoptent une approche fondée sur le marché [27, 11, 21] qui modélise l'équilibrage de charge comme un jeu non coopératif afin d'optimiser des métriques centrées sur l'utilisateur plutôt que des métriques centrées sur le système, telles que le temps d'achèvement global que nous utilisons dans cet article.

Contrairement à notre travail, [24] assignent les tâches à des coalitions d'agents car ils ne peuvent pas réaliser les tâches seuls. 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. [8] ont étudié l'allocation multi-agents de ressources mais leur travail se limite à l'affectation d'une seule ressource par agent. Nous supposons ici qu'un lot de tâches peut être assigné à chaque agent. Dans la continuité de [2, 10], [22] proposent des méthodes potentiellement distribuables pour l'allocation de ressources qui s'appuient sur la négociation multi-agents. Même si la topologie du réseau n'entre pas dans le cadre de notre travail (nous partons du principe que les agents sont pleinement connectés), nous allons un peu plus loin en décentralisant effectivement la négociation et en remettant en jeu cette allocation pendant le traitement des tâches.

Il est bien connu que le traitement de jeux de données réels via le déploiement distribué du patron de conception MapReduce comporte souvent des biais de données qui peuvent conduire à un déséquilibre des charges [17]. En particulier, le biais de partitionnement se produit lorsqu'un *reducer* traite un plus grand nombre de clés que les autres. Puisque le *job* se termine lorsque toutes les tâches *reduce* sont terminées, le temps d'exécution du *job* est pénalisé par le *reducer* le plus chargé. Ce biais de données est abordé par [7, 19] à l'aide d'un paramétrage dépendant de connaissances préalables sur des données et sur l'environnent d'exécution. Nous abordons cette question à l'aide d'une réallocation de tâches dynamique et adaptative qui est concurrente à la consommation des tâches. Ainsi, la réallocation s'adapte au traitement des données. Cela nous permet d'aborder 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.

3 Allocation de tâches situées

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

Définition 1 (MASTA)**.** *Un problème d'allocation multi-agents de tâches situées de taille* (k, m, n) *avec* $k \geq 1$, $m \geq 2$ *et* $n \geq 1$ *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 ;*
- $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ *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. Plus une tâche est locale, moins elle coûte chère :*

$$\forall i, j \in \mathcal{A}, d(\tau, l(i)) > d(\tau, l(j)) \Rightarrow c_i(\tau, l(i)) \leq c(\tau, l(j)) \quad (1)$$

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 $\sum_{node \in Node} d(\tau, node)$. On dit que τ est locale (resp. semi-locale, distante) pour l'agent i si $d_i(\tau) = d_\tau$ (resp. $d_i(\tau) < d_\tau$, $d_i(\tau) = 0$).

Étant donné 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 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 :*

$$\cup_{i \in \mathcal{A}} P(i) = \mathcal{T} \quad (2)$$

$$\forall i \in \mathcal{A}, \forall j \in \mathcal{A} \setminus \{i\}, P(i) \cap P(j) = \emptyset \quad (3)$$

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

$$w_i(P) = \sum_{\tau \in P(i)} c_i(\tau) \quad (4)$$

Le makespan de P est défini par :

$$C_{max}(P) = \max\{w_i(P) \mid i \in \mathcal{A}\} \quad (5)$$

Les agents réalisent des délégations de tâches qui sont socialement rationnelles afin d'améliorer l'allocation.

Définition 3 (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 :*

$$\forall k \in \mathcal{A} \setminus \{i, j\}, P'(k) = P(k) \quad (6)$$

$$P'(i) = P(i) \setminus \{\tau\} \wedge P'(j) = P(j) \cup \{\tau\} \quad (7)$$

La délégation est socialement rationnelle si et seulement si :

$$w_j(P) + c_j(\tau) < w_i(P) \quad (8)$$

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

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

$$\Gamma_i(P) = \{\tau \in P(i) \mid w_j(P) + c_j(\tau) < w_i(P)\} \quad (9)$$

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

Propriété 1. *On peut toujours aboutir à une allocation stable à partir d'une allocation non stable en utilisant un nombre fini de délégations socialement rationnelles.*

Preuve 1. *Soit P une allocation de tâches et $W_P =< w_{i_1}, \ldots, w_{i_m} >$ le vecteur des charges par ordre décroissant où w_{i_r} dénote la r^{ieme} charge la plus élevée. Si P n'est pas stable, il existe une délégation socialement rationnelle δ qui mène à $P' = \delta(P)$. Formellement,*

$$\exists i, j \in \mathcal{A}, max\big(w_i(P'), w_j(P')\big) < max\big(w_i(P), w_j(P)\big) \wedge \forall k \in \mathcal{A} \setminus \{i, j\}, w_k(P) = w_k(P') \quad (10)$$

Par conséquent, $W_{P'} < W_P$ selon l'ordre lexicographique. Formellement,

$$\exists r \in [1, m]\ \forall r' < r, W_{P'}(r') = W_P(r') \wedge W_{P'}(r) < W_P(r) \quad (11)$$

Puisqu'il y a un nombre fini d'allocations et que l'ordre est strict, il y a un nombre fini de délégations socialement rationnelles.

Voici un exemple pour illustrer les différentes définitions.

Exemple 1. *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 1. Soient les allocations Pmks et P telles que*

	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

TABLE 1 – Localisations et coûts des tâches

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}(Pmks) $= 35$ (w_1(Pmks) $= 33$ et w_2(Pmks) $= 35$). Ce n'est pas le cas de P puisque $w_1(P) = 46$, $w_2(P) = 27$, et donc $C_{max}(P) = 46$. De plus, $\Gamma_1(P) = \{\tau_2, \tau_6\}$ et $\Gamma_2(P) = \emptyset$.

Soit $P' = \delta_1(P)$ l'allocation obtenue par la délégation δ_1 de la tâche τ_6 par l'agent 1 à l'agent 2. 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 car $\Gamma_2(P') = \{\tau_1\}$.

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 ($C_{max}(P'') > C_{max}$(Pmks)), P'' est stable car $\Gamma_1(P'') = \Gamma_2(P'') = \emptyset$.

4 Processus de négociation

Cette section présente le processus de négociation qui est itéré et 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* d'un nouveau problème MASTA. En effet, l'accomplissement d'une tâche est un évènement disruptif qui modifie les paramètres du problème et l'allocation des tâches.

Définition 4 (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 :*

$$\mathcal{T}' = \mathcal{T} \setminus \{\tau\} \tag{12}$$

$$P'(i) = P(i) \setminus \{\tau\} \tag{13}$$

$$\forall j \in \mathcal{A} \setminus \{i\}, P'(j) = P(j) \tag{14}$$

À l'évidence, la consommation d'une tâche diminue le *makespan*. La séquence des consommations, qui retirent peu à peu toutes les tâches de l'allocation initiale jusqu'à l'allocation vide $\perp$, consiste en une itération de problèmes MASTA.

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 [25]. Il y a 3 étapes de décision durant une négociation : (a) l'initiateur de l'enchère choisit une tâche selon une des stratégies décrites dans la section 5, (b) chaque agent propose/refuse de prendre la tâche selon que la délégation est socialement rationnelle ou non, et (c) l'initiateur sélectionne le vainqueur de l'enchère, e.g. l'enchérisseur qui a la plus petite charge de travail. Plusieurs négociations impliquant différents groupes d'agents peuvent se dérouler simultanément et les agents peuvent participer à plusieurs enchères en même temps. La concurrence des délégations de tâches améliore la réactivité de la réallocation des tâches.

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 :

$$\mathcal{B}_i(P) = \langle w_1^i(P), \ldots, w_{i-1}^i(P), w_{i+1}^i(P), \ldots, w_m^i(P)\rangle \tag{15}$$

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.

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 s'appuie sur sa base de croyances $\mathcal{B}_i(P)$. Formellement,

$$\Gamma_i^{\mathcal{B}}(P) = \{\tau \in P(i) \mid w_j^i(P) + c_j(\tau) < w_i(P)\}. \tag{16}$$

Lorsqu'un agent souhaite initier une enchère, le calcul du *makespan* local s'appuie sur sa base de croyances, éventuellement erronée. C'est le prix à payer pour la décentralisation. Cependant, un agent informe ses pairs 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*. En adoptant une approche conservatrice [23], nous nous assurons qu'une négociation couronnée de succès améliore strictement l'allocation de tâches et donc que le processus de négociation converge.

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 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 (e.g. après P'_{stable}).

5 Stratégie situationnelle

Au cours du processus, les négociations et les consommations de tâches sont concurrentes. La stratégie d'un agent consiste à choisir la prochaine délégation potentiellement socialement rationnelle et la prochaine tâche à exécu-

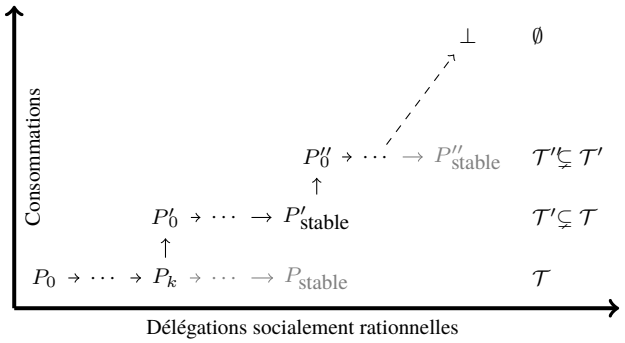

FIGURE 1 – Consommations de tâches (arcs verticaux) et délégations de tâches (arcs horizontaux) concurrentes.

ter.

Stratégie agnostique vis-à-vis de la localité. Dans une première approche, un agent i peut exécuter les tâches les plus grandes de son propre lot et négocier les plus petites parmi les délégations potentiellement socialement rationnelles. Formellement,

$$\overset{>}{\tau} = \underset{\tau \in P(i)}{\text{argmax}}\{c_i(\tau)\} \tag{17}$$

$$\overset{<}{\tau} = \underset{\tau \in \Gamma_i^{\mathcal{B}}(P)}{\text{argmin}}\{c_i(\tau)\} \tag{18}$$

où $\overset{>}{\tau}$ (resp. $\overset{<}{\tau}$) représente la prochaine tâche à exécuter (resp. à négocier). Cette stratégie nécessite que l'agent trie son lot de tâches selon leurs coûts. Cependant, acquérir une ressource consomme du temps et représente un coût supplémentaire lors de l'exécution.

Afin de mesurer la localité des tâches, on définit le ratio de possession d'une tâche par un agent i comme le ratio entre le nombre de ressources locales à l'agent pour cette tâche et le nombre total de ressources de la tâche :

$$o_i(\tau) = \frac{d_i(\tau)}{d_\tau} \tag{19}$$

Le ratio maximal de possession de τ est :

$$\hat{\text{o}}(\tau) = max_{i \in \mathcal{A}}\{o_i(\tau)\} \tag{20}$$

Stratégie situationnelle. Intuitivement, un agent devrait d'abord exécuter les tâches qui peuvent coûter plus pour les autres que pour lui-même, et déléguer les tâches qui peuvent coûter moins pour les autres. Selon cette stratégie, un agent exécute d'abord les grandes tâches locales et négocie d'abord les grandes tâches distantes, en fonction des ses connaissances et croyances locales, i.e. le coût de chaque tâche et son ratio de possession. Cette stratégie utilise une structure de données appelée lot par possession.

Le **lot par possession** de l'agent i (voir figure 2) est partagé en trois paquets selon le ratio de possession de l'agent pour ces tâches.

1. *Le paquet local* (noté MB) contient les tâches dont l'agent possède au moins une ressource et dont il est le plus gros propriétaire. Formellement,

   $$\forall \tau \in MB, o_i(\tau) \neq 0 \wedge o_i(\tau) = \hat{\text{o}}(\tau) \tag{21}$$

   Dans MB, les tâches sont triées par ordre décroissant de coût (cf. gauche de la figure 2).
2. *Le paquet semi-local* (noté IB) contient les tâches partiellement locales. Formellement,

   $$\forall \tau \in IB, 0 < o_i(\tau) < \hat{\text{o}}(\tau). \tag{22}$$

   Dans IB, les tâches sont triées par ordre décroissant de ratio de possession et les tâches ayant le même ratio de possession sont triées par ordre décroissant de coût (cf. centre de la figure 2).
3. *Le paquet distant* (noté DB) contient les tâches distantes. Formellement,

   $$\forall \tau \in DB, o_i(\tau) = 0. \tag{23}$$

   Dans DB, les tâches sont triées par ordre croissant de coût. (cf. droite de la figure 2).

Quand un agent recherche une tâche à exécuter, il commence par le début du paquet local, i.e. par la tâche locale la plus grande. Quand un agent cherche une tâche à négocier, il commence par la fin du paquet distant (i.e. la plus grande tâche distante) et il sélectionne la première qui constitue une délégation potentiellement socialement rationnelle selon ses croyances, $\mathcal{B}_i(P)$. Formellement,

— Un agent cherche une tâche à exécuter $\overset{>}{\tau} \in MB$ telle que

$$\forall \tau \in MB \setminus \{\overset{>}{\tau}\},\ c_i(\overset{>}{\tau}) \geq c_i(\tau). \tag{24}$$

Si $MB = \emptyset$, il cherche $\overset{>}{\tau} \in IB$ telle que

$$\begin{aligned} &\forall \tau \in IB \setminus \{\overset{>}{\tau}\}, \\ &o_i(\overset{>}{\tau}) > o_i(\tau) \vee \\ &\big(o_i(\overset{>}{\tau}) = o_i(\tau) \wedge c_i(\overset{>}{\tau}) \geq c_i(\tau)\big). \end{aligned} \tag{25}$$

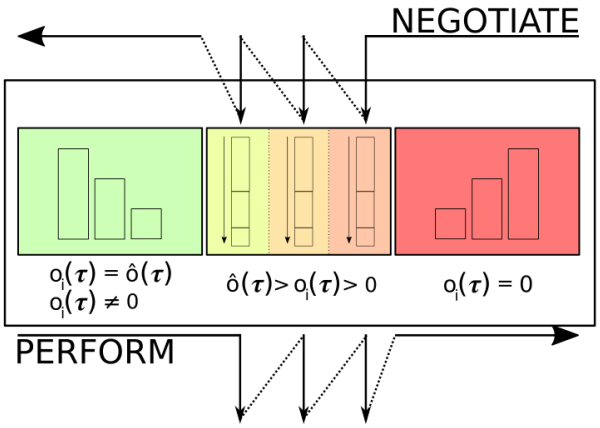

FIGURE 2 – Le lot par possession contient le paquet local (à gauche), le paquet semi-local (au centre) et le paquet distant (à droite). Les tâches sont représentées par des rectangles dont la taille est proportionnelle au coût de la tâche. Les flèches désignent l'ordre de parcours selon lequel un agent va chercher une tâche à exécuter/négocier.

Si $IB = \emptyset$, il cherche $\overset{>}{\tau} \in DB$ telle que

$$\forall \tau \in DB \setminus \{\overset{>}{\tau}\},\ c_i(\overset{>}{\tau}) \leq c_i(\tau). \quad (26)$$

Finalement, si $MB = IB = DB = \emptyset$, l'agent n'a aucune tâche à exécuter.

— Un agent cherche une tâche à négocier $\overset{<}{\tau} \in DB$ telle que

$$\forall \tau \in \big(DB \cap \Gamma_i^{\mathcal{B}}(P)\big) \setminus \{\overset{<}{\tau}\},\ c_i(\overset{<}{\tau}) \geq c_i(\tau). \quad (27)$$

Si $DB = \emptyset$, il cherche $\overset{<}{\tau} \in IB$ telle que

$$\forall \tau \in \big(IB \cap \Gamma_i^{\mathcal{B}}(P)\big) \setminus \{\overset{<}{\tau}\},\ o_i(\overset{<}{\tau}) < o_i(\tau) \vee \big(o_i(\overset{<}{\tau}) = o_i(\tau) \wedge c_i(\overset{<}{\tau}) \geq c_i(\tau)\big). \quad (28)$$

Si $IB = \emptyset$, il cherche $\overset{<}{\tau} \in MB$ telle que

$$\forall \tau \in \big(MB \cap \Gamma_i^{\mathcal{B}}(P)\big) \setminus \{\overset{<}{\tau}\},\ c_i(\overset{<}{\tau}) \leq c_i(\tau). \quad (29)$$

Finalement, si l'agent ne trouve pas une telle tâche, il arrête d'initier des enchères.

Il est important de remarquer que l'ordre leximax sur le paquet semi-local est conforme au principe selon lequel on exécute d'abord les grandes tâches locales et on négocie d'abord les grandes tâches distantes. Quand un agent explore le paquet semi-local, il commence par les tâches qui ont un ratio de possession élevé afin de les exécuter, et par celles qui ont un ratio de possession faible afin de les négocier. Quand un agent examine les tâches qui ont le même ratio de possession, il le fait toujours dans le même ordre : des plus coûteuses vers les moins coûteuses (cf. figure 2). Donc, quand un agent explore le paquet semi-local à la recherche d'une tâche à exécuter, il considère la plus locale et plus coûteuse ; et quand un agent explore le paquet semi-local à la recherche d'une tâche à négocier, il considère la moins locale et plus coûteuse.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
$o_1(\tau_k)$	1	0	0.33	0.33	0.2	0.85	0
$o_2(\tau_k)$	0	1	0.66	0.66	0.8	0.14	1

TABLE 2 – Ratios de possession pour l'exemple 2

Exemple 2 (Lot par possession)**.** *Reprenons l'exemple 1 où nous supposons qu'une ressource distante est deux fois plus coûteuse qu'une ressource locale. Formellement,*

$$c_i(\tau) = d_i(\tau) + 2\big(d_\tau - d_i(\tau)\big) = 2d_\tau - d_i(\tau) \quad (30)$$

Le tableau 2 donne le ratio de possession de chaque agent pour chaque tâche. Considérons l'allocation où toutes les tâches sont affectées à l'agent 1*. Selon la stratégie agnostique, l'agent* 1 *ne fait pas de différence entre* τ_2 *et* τ_6 *pour une délégation puisque ces tâches ont le même coût pour lui. Selon la stratégie situationnelle, l'agent* 1 *dont le lot par possession est donné figure 3 considère :*

- *l'exécution des tâches* τ_6, τ_1, τ_4, τ_3, τ_5, τ_2, *puis* τ_7 *;*
- *la négociation des tâches* τ_7, τ_2, τ_5, τ_4, τ_3, τ_1, *puis* τ_6.

La stratégie situationnelle amène l'agent 1 *à exécuter* τ_6 *et à négocier* τ_7 *afin de diminuer le makespan, puisque* τ_6 *est en grande partie locale et* τ_7 *est distante. En particulier, l'agent* 1 *exécute* τ_6 *avant* τ_1 *car* τ_6 *est plus coûteuse (cf. table 1).*

Nous soulignons que les tâches dans le paquet semi-local sont triées d'abord selon le ratio de possession puis selon leur coût : que l'agent 1 *cherche à négocier ou à exécuter une tâche, il considère celles qui ont le même ratio maximal de possession et dans le même ordre (par ex.* τ_4*, puis* τ_3*).*

6 Application pratique

Comme application pratique, nous considérons le déploiement distribué du patron de conception MapReduce pour le traitement de jeux de

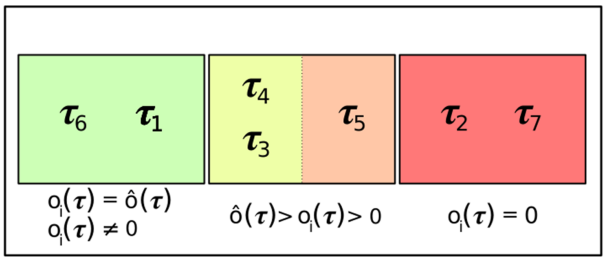

FIGURE 3 – Lot par possession de l'agent 1

données de grandes tailles sur une grappe de serveurs [9]. [17] identifient plusieurs biais qui pénalisent le temps nécessaire au traitement des données. En particulier, le biais de partitionnement se produit quand un *reducer* doit traiter un plus grand nombre de clés (et donc de tâches) que les autres. Ce biais produit un déséquilibre dans les charges de travail des *reducers*. La stratégie situationnelle permet aux agents d'équilibrer les charges et ainsi d'améliorer le temps d'exécution.

Nous avons déployé MapReduce à l'aide d'un système distribué multi-agents. Notre prototype [2] a été implémenté en Scala avec la boîte à outils Akka [3] adaptée 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 : (a) le délai de transmission des messages est arbitraire mais non négligeable, (b) l'ordre d'émission/réception des messages est identique par pair émetteur-récepteur et (c) les messages peuvent être perdus. Dans un tel système, les messages sont livrés 0 ou 1 fois. C'est la raison pour laquelle nous avons intégré un mécanisme d'interruptions temporelles dans le protocole d'interaction.

Nos expériences [4] utilisent un jeu de données de 8 Gio ($82,283$ clés) qui a été généré de telle sorte que l'allocation initiale des tâches (cf. figure 4) soit déséquilibrée et de manière à pouvoir vérifier que la proximité entre les données et les nœuds de traitement a un impact sur le *makespan* et donc sur le temps d'exécution. L'allocation initiale des tâches est le résultat de la phase de partitionnement du processus MapReduce. En d'autres termes, quand il n'y a pas de négociation, notre système a le même comportement que Hadoop.

2. https://github.com/cristal-smac/mas4data
3. http://akka.io
4. Nos expériences ont été réalisées sur 16 PCs quadricœurs Intel(R) i7 avec 16GB RAM chacun.

Nous comparons, à partir de 10 exécutions pour chacune des (ré)allocations, les temps d'exécutions médians quand les agents adoptent une stratégie agnostique ou situationnelle. Nous observons que la stratégie situationnelle améliore significativement le temps d'exécution, d'environ -7.6%. Pour comparaison, le temps d'exécution sans remise en cause du partitionnement par défaut d'Hadoop est de $853s$ (environ $+100\%$). Nous en déduisons que le coût de la négociation peut être négligé au regard du gain obtenu par équilibrage des charges. De plus, la négociation permet de diminuer le temps d'exécution, tout particulièrement avec la stratégie situationnelle.

En raison du non-déterminisme de l'exécution distribuée, nous présentons un *run* typique. La figure 4 compare l'allocation de tâches quand toutes les tâches ont été traitées, qu'elles aient été négociées ou pas entre les 16 agents [5], selon les stratégies agnostique ou situationnelle. Les deux stratégies sont comparées *ex-post*. Nous remarquons que le *makespan* de l'allocation initiale vaut approximativement $3.3\ 10^8$, il est de $2.5\ 10^8$ (-24%) en cas de négociation avec la stratégie agnostique locale et de $2\ 10^8$ (-30.7%) avec la stratégie situationnelle. Nous en déduisons que la négociation, en particulier avec la stratégie situationnelle, permet d'améliorer l'équilibre des charges.

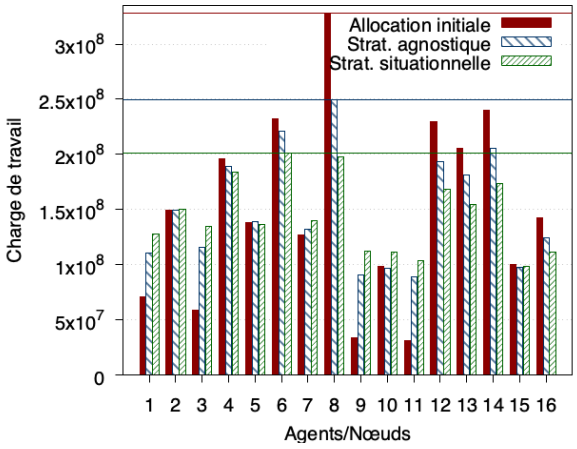

FIGURE 4 – Allocation initiale des tâches (Hadoop) et allocation *ex-post* avec les stratégies agnostique et situationnelle

La figure 5 présente le nombre de tâches t telles que $\alpha \leq o_i(t) < \alpha + 0.1$ quand l'agent i réalise t. L'allocation initiale de tâches est mal équilibrée

5. Une analyse empirique préliminaire montre qu'il n'y a pas de valeur ajoutée à avoir plus d'un *reducer* par nœud/disque.

car il y a plus de $4.6\ 10^4$ tâches pour lesquelles $o_i(\tau) = 0$. Nous observons que la stratégie situationnelle favorise le traitement des tâches qui sont les plus locales, i.e. $o_i(\tau) \geq 0.5$. Ce n'est pas le cas de la stratégie agnostique locale. Par exemple, il y a 171 tâches qui ont été traitées par un agent qui possède 60% des *chunks* (i.e. des ressources) avec la stratégie situationnelle, mais aucune avec la stratégie agnostique. Puisqu'elle favorise l'exécution des tâches par les nœuds qui sont les plus proches des chunks, la stratégie situationnelle améliore l'équilibre des charges et diminue le temps d'exécution.

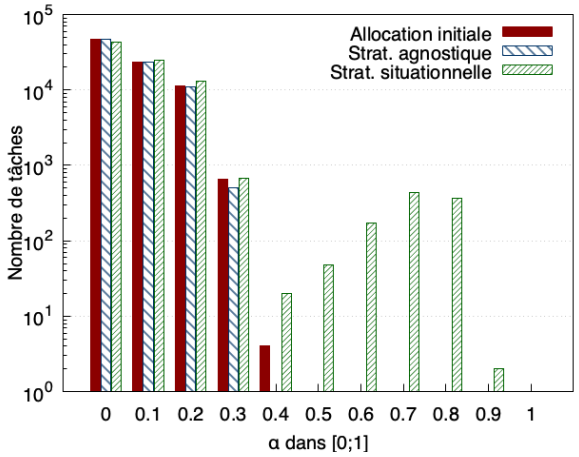

FIGURE 5 – Nombre de tâches par ratio de possession pour l'agent exécutant (échelle logarithmique)

7 Conclusion

Dans cet article, nous avons formalisé le problème de l'allocation multi-agents de tâches situées et nous avons proposé une stratégie situationnelle pour traiter le problème de l'équilibrage des charges dans les systèmes distribués. Cette stratégie adopte une approche fondée sur le marché qui nous permet de décentraliser l'équilibrage de charge pour minimiser le *makespan*, i.e. le temps nécessaire pour achever la dernière tâche. Au cours du processus, les négociations de tâches permettent aux agents qui utilisent la stratégie situationnelle de réallouer les tâches en même temps qu'ils en consomment. Cette stratégie permet également à un agent de déterminer la prochaine délégation socialement rationnelle et la prochaine tâche à traiter. Conformément à leur propre base de croyances et à leurs connaissances, les agents traitent en priorité les grandes tâches locales et négocient en priorité les grandes tâches distantes. Afin de valider notre approche, nous avons développé un prototype dans lequel les agents négocient les tâches de *reduce* du patron de conception MapReduce dans une configuration distribuée. Nos résultats expérimentaux montrent que la stratégie situationnelle proposée améliore l'équilibre de charge et donc le temps d'exécution pour de telles applications.

Actuellement, nous évaluons notre prototype dans différentes configurations matérielles, avec de nombreuses expérimentations qui se basent sur des jeux de données réels. Certaines configurations nous amènent à considérer que nous devons étendre notre stratégie pour négocier (a) des échanges de tâches afin d'améliorer le *makespan* des allocations stables, et (b) des lots de tâches pour accélérer les négociations.

Remerciements. Ce travail est soutenu par l'AAP ULille « Internationalisation Actions bilatérales ». 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 AAMAS*, pages 981–988, 2010.

[2] Martin R. Andersson and Tuomas W. Sandholm. Contract types for satisficing task allocation : Ii experimental results. In *Proc. of the AAAI spring symposium : Satisficing models*, pages 1–7, 1998.

[3] Gamal Attiya and Yskandar Hamam. Task allocation for maximizing reliability of distributed systems : A simulated annealing approach. *Journal of Parallel and Distributed Computing*, 66(10) :1259–1266, 2006.

[4] Quentin Baert, Anne-Cécile Caron, Maxime Morge, and Jean-Christophe Routier. Stratégie de découpe de tâche pour le traitement de données massives. In *Actes des JFSMA*, pages 65–74, 2017.

[5] Quentin Baert, Anne-Cécile Caron, Maxime Morge, Jean-Christophe Routier, and Stathis Kostas. Adaptive multi-agent system for situated task allocation. In *Proc. of AAMAS*, pages 1790–1791, 2019.

[6] Bo Chen, Chris N. Potts, and Gerhard J Woeginger. *Handbook of combinatorial optimization*, chapter A Review of Machine Scheduling : Complexity, Algorithms and Approximability, pages 1493–1641. Springer, 1998.

[7] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. SAMR : A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In *International Conference on Computer and Information Technology*, pages 2736–2743. IEEE, 2010.

[8] Anastasia Damamme, Aurélie Beynier, Yann Chevaleyre, and Nicolas Maudet. The Power of Swap Deals in Distributed Resource Allocation. In *Proc. of AAMAS*, pages 625–633, 2015.

[9] J. Dean and S. Ghemawat. MapReduce : Simplified data processing on large clusters. In *Symposium on Operating Systems Design and Implementation*, pages 137–150, 2004.

[10] Ulle Endriss, Nicolas Maudet, Fariba Sadri, and Francesca Toni. Negotiating socially optimal allocations of resources. *JAIR*, 25(1) :315–348, 2006.

[11] Saurabh Kumar Garg, Srikumar Venugopal, James Broberg, and Rajkumar Buyya. Double auction-inspired meta-scheduling of parallel applications on global grids. *Journal of Parallel and Distributed Computing*, 73(4) :450–464, 2013.

[12] A. M. A. Hariri and N. Potts, Chris. Heuristics for scheduling unrelated parallel machines. *Computers & operations research*, 18(3) :323–331, 1991.

[13] Ellis Horowitz and Sartaj Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *JACM*, 23(2) :317–327, 1976.

[14] Oscar H Ibarra and Chul E Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *JACM*, 24(2) :280–289, 1977.

[15] 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.

[16] Qin-Ma Kang, Hong He, Hui-Min Song, and Rong Deng. Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization. *Journal of Systems and Software*, 83(11) :2165–2174, 2010.

[17] YongChul Kwon, Kai Ren, Magdalena Balazinska, and Bill Howe. Managing skew in Hadoop. *IEEE Data Eng. Bull.*, 36(1) :24–33, 2013.

[18] Jan Karel Lenstra, David B Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3) :259–271, 1990.

[19] Miguel Liroz-Gistau, Reza Akbarinia, and Patrick Valduriez. FP-Hadoop : efficient execution of parallel jobs over skewed data. *VLDB Endowment*, 8(12) :1856–1859, 2015.

[20] Silvano Martello, François Soumis, and Paolo Toth. Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete applied mathematics*, 75(2) :169–188, 1997.

[21] Amro Najjar, Olivier Boissier, and Gauthier Picard. Négociation adaptative pour l'acceptabilité des services d'un fournisseur SaaS. In *Actes des JFSMA*, pages 85–94, 2017.

[22] Antoine Nongaillard and Philippe Mathieu. Reallocation problems in agent societies : A local mechanism to maximize social welfare. *JASSS*, 14(3) :21, 2011.

[23] Michael Schillo, Christian Kray, and Klaus Fischer. The eager bidder problem : a fundamental problem of DAI and selected solutions. In *Proc. of AAMAS*, pages 599–606. ACM, 2002.

[24] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artif. Intell.*, 101(1-2) :165–200, 1998.

[25] Reid G. Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *IEEE Transactions on computers*, 29(12) :1104–1113, December 1980.

[26] Joanna Turner, Qinggang Meng, Gerald Schaefer, and Andrea Soltoggio. Distributed strategy adaptation with a prediction function in multi-agent task allocation. In *Proc. of AAMAS*, pages 739–747, 2018.

[27] William E Walsh and Michael P Wellman. A market protocol for decentralized task allocation. In *Proc. of ICMAS*, pages 325–332, 1998.

[28] Peng-Yeng Yin, Shiuh-Sheng Yu, Pei-Pei Wang, and Yi-Te Wang. Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. *Journal of Systems and Software*, 80(5) :724–735, 2007.