Task Bundle Delegation for Reducing the Flowtime

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

Task Bundle Delegation for Reducing the Flowtime

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

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
{ellie.beauprez, anne-cecile.caron, maxime.morge,
jean-christophe.routier}@univ-lille.fr

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

Keywords: Multi-Agents Systems · Distributed Problem Solving · Agent-based Negotiation.

1 Introduction

The problem of efficient task assignment among executing entities is common to many real-world applications for logistics [13, 15], collective robotics [7, 22], distributed systems [20, 12], or more recently Big Data [1]. In particular, Data Science, which involves the processing of large volumes of data which requires distributed file systems and parallel programming, challenges distributed computing with regard to task allocation and load-balancing. This paper is concerned with a class of practical applications where (a) some concurrent jobs (sets of tasks) must be performed as soon as possible, (b) the resources (e.g. data) required to successfully execute a task are distributed among nodes. Here we consider the most prominent distributed data processing model for tackling vast amount of data on commodity clusters, i.e. the MapReduce design pattern [24]. Jobs are composed of tasks executed by the different nodes where the resources are distributed. Since several resources are necessary to perform a task, its execution requires fetching some of these resources from other nodes, thus incurring an extra time cost for its execution.

Many works adopt the multi-agent paradigm to address the problem of task reallocation and load-balancing in distributed systems [12]. The individual-based approach allows the decentralization of heuristics to scale-up the resolution of scheduling problem despite of the combinatory explosion. Furthermore, due to their inherent reactive

nature, the multi-agent methods for task reallocation are adaptive to the inaccurate estimation of task execution time and some disruptive phenomena (task consumption, job release, slowing down nodes, etc.). Most of these works adopt the market-oriented approach which models problems as non-cooperative games [23, 7], eventually with machine learning techniques which assume past experiences [22, 20]. By contrast, we suppose as [1] that: (a) the agents are cooperative, i.e. they have a partial and local perception of the task allocation but they share the same goal, and (b) neither generalizable predictive patterns nor prior model of the data/environment are available since it is not the case for the class of practical applications we are concerned with. We go further here by considering several concurrent jobs composed of tasks. Each task can be executed by a single agent, all of them are competent. Agents want to minimize the mean flowtime of jobs. The main difficulty lies in the formulation of complex systems for the reassignment of tasks-workers which are decentralized and adaptive, i.e. the design of individual and asynchronous behaviours that lead to the emergence of feasible allocations combining the objectives of the task requesters.

We propose a strategy that decides which bilateral reallocation is suggested or accepted. Based on peer modelling, the strategy determines the agent behaviour in the negotiation protocol. The offer strategy selects a potential delegation, i.e. an offer bundle and a respondant. The acceptability rule determines whether the agent accepts or rejects all or part of this delegation. Specifically, our contributions are as follows:

1. We formalize the multi-agent task allocation problem where concurrent jobs are composed of situated tasks with different costs depending on the location of the resources.
2. We propose a strategy that continuously identifies bottleneck agents and opportunities within unbalanced allocations to trigger concurrent and bilateral negotiations in order to delegate or swap tasks.
3. We conduct extensive experiments that show that our method reaches a flowtime close to the one reached by the the classical heuristic and significantly reduces the rescheduling time.

This paper is a extended version of [2].

1. We generalize the notion of delegation to consider any bilateral reallocation (delegation or swap of several tasks).
2. We redefine the acceptability criterion of the bilateral reallocations in order to reduce the rescheduling time and the mean flowtime reached by our strategy.
3. We quantitatively compare the performance of our strategy with the performance of a DCOP resolution method.

After an overview of related works in Section 2, we formalize the multi-agent situated task allocation problem in Section 3. Section 4 describes the consumption/delegation operations and the negotiation process. Section 5 specifies how agents choose which tasks to negotiate and with whom. Our empirical evaluation is described in Section 6. Section 7 summarizes our contribution and future work.

	Resources	**Tasks**	**Workers**	**Objective**	**Dynamics**	**Decentralized**	**Approach/ Technique**
(Kuhn-Munkres, 1955) [13]	—	n single-worker	R_n single-task	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✗	✗	LP
(SPT, 1967) [8]	—	n single-worker	P_m multi-task	$\mathrm{C}(\overrightarrow{\mathrm{A}})$	✗	✗	Heuristic
(Bruno et al., 1974) [5]	—	n single-worker	R_m multi-task	$\mathrm{C}(\overrightarrow{\mathrm{A}})$	✗	✗	LP
(Shehory et Krause, 1998) [21]	—	n multi-worker	R_m single-task	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	Coalition
(GPGP, 2004) [14]	consumables or not	n single-worker	R_m multi-task	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	Team
(Turner et al., 2018) [22]	scarce	n single-worker	R_m online multi-task	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	CBBA + ML
(Li et al., 2014) [15]	—	n single-worker	P_m multi-task	$\mathrm{C}(\overrightarrow{\mathrm{A}})$ $\oplus\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	DCOP
(Schaerf et al., 1995) [20]	—	n single-worker	P_m multi-task	$\mathrm{W}(\overrightarrow{\mathrm{A}})$	✓	✓	MARL
(MASTA, 2021) [1]	transferables duplicables	n single-worker	R_m multi-task	$\mathrm{C}_{max}(\overrightarrow{\mathrm{A}})$	✓	✓	Team
(SMASTA+, 2021) [2]	transferables duplicables	n multi-worker	R_m multi-task	$\mathrm{C}(\overrightarrow{\mathrm{A}})$	✓	✓	Team

Table 1: Analysis grid of methods for task assignment (at top) or reassignment (at bottom)

2 Related work

Table 1 summarizes all the works discussed here according to our analysis grid. The left-hand side of the table shows the problems which are addressed, i.e. their ingredients (resources, tasks, workers and objectives). The right-hand side reveals the features of these methods and the techniques used. While the upper part of the table contains some classical task assignment methods, the lower part presents some dynamic and decentralized reassignment methods.

Scheduling theory [6] includes off-line methods for solving various problems of task assignment among workers. For instance, the Kuhn-Munkres algorithm, also called the Hungarian method, minimizes the total cost (denoted $\mathrm{W}(\overrightarrow{\mathrm{A}})$) for n tasks and n workers [13]. *Shortest processing time first* (SPT) is a very simple method that minimizes the flowtime of n single-worker tasks with one multi-task worker [8]. This result generalizes to the problem with m multi-task workers if the cost of tasks is identical from one worker to another (denoted P_m). The scheduling problem, which consists in minimizing the total delay (denoted $\mathrm{C}(\overrightarrow{\mathrm{A}})$) with m multi-task workers and n single-worker tasks whose costs depend on the worker (denoted R_m), can be formalized by a linear-program (LP). This problem reduces to a weighted matching problem in a bipartite graph with n tasks and $n \times m$ positions. This problem is polynomial [11]. Based on the Ford-Fulkerson algorithm, the complexity of the algorithm described by [5] is $O(max(mn^2, n^3))$. These approaches are not suitable for task reassignment in dis-

tributed systems where decentralization and adaptability are required. Indeed, global control is a performance bottleneck, since it must constantly collect information about the state of the system. By contrast, our agents make local decisions on an existing allocation with the aim to improve the load-balancing. Moreover, classical scheduling problems are static. The inaccurate estimation of the task execution time and some disruptive phenomena (task consumption, job release, slowing down nodes, etc.) may require major modifications in the existing allocation to stay optimal. Furthermore, agents can operate in dynamic environments that change over time.

The multi-agent paradigm is particularly suitable for the design and implementation of distributed and adaptive mechanisms for the reassignment of tasks-workers [12]. The existing models differ due to the nature of the tasks and the agents, whether they represent workers or task requesters. Coalition formation is justified if a task requires more than one worker or if its cost decreases with the number of assigned workers. For instance, Shehory and Kraus propose decentralized, greedy and anytime algorithms for assigning multi-worker tasks with precedence constraints to some workers with heterogeneous capabilities/efficiencies. [21]. Similarly to a coalition, a team aims at maximizing an overall objective function rather than the individual welfares. However, a team performs single-worker tasks. For instance, Lesser et al. propose a domain-independent coordination framework with a hierarchical task network representation for resource allocation and task assignment/scheduling [14]. The main objective of *Generalized Partial Global Planning* (GPGP) is the maximization of the combined utility accrued by the group of agents as a result of successful completion of its high-level goals. GPGP adopts a planning-oriented approach of coordination which assumes that the effort required for coordination (reasoning and communication) is negligible compared to the tasks execution time. Inspired by economic theories, the market-oriented approach models distributed planning problems as the search of an equilibrium for a non-cooperative game [23]. The agents delegate/swap tasks. Contrary to a team, a marketplace assumes that the constraints and objectives are fully distributed. Among the market-oriented methods, we distinguish three families.

DCOP. The reassignment problems can be formalized as *Distributed Constraint Optimization Problems* (DCOP). Many resolution methods have been developed for finding an optimal solution to a DCOP which is an NP-hard problem (see a recent survey [9]). The main difficulty in applying these methods for task reassignment lies in the representation of a real-world problem as a DCOP, or even several COP subproblems, since it requires expertise in the resolution method (e.g. [15]).

CBBA. *Consensus Based Bundle Algorithm* [7] is a multi-agent assignment method that consists of: (a) selecting the negotiated tasks; (b) determining the winner of these negotiations. In the same line, Turner et al. study the continuous task assignment for a multi-robot team in order to maximise the throughput before running out of fuel [22]. Thanks to machine learning (ML), they propose a prediction mechanism that uses past experience to select which task allocation strategy yields the optimal global task allocation.

MARL. The reassignment problems can also be formalized as Markov decision processes [4], in particular Decentralized Partially Observed Markov Decision Process (Dec-POMDP). The optimization of a Dec-POMDP with a finite horizon is a NEXP-

TIME problem. Approximate resolution methods can only be applied to small problem instances, they do not scale up. Beyond these off-line planning methods, *Multi-Agent Reinforcement Learning* (MARL) requires a perfect knowledge of the environment and a learning phase [20].

Conversely, we consider neither generalizable predictive patterns nor prior model of the data/environment are available since it is not the case for the class of practical applications we are concerned with. For instance, Baert et al. have in [1] an egalitarian objective which is the minimization of the makespan (denoted $C_{max}(\overrightarrow{A})$). We consider here the problem of coordinating decisions between agents to find a globally optimal solution for a multi-objective function. Agents want to minimize the mean flowtime of several concurrent jobs, each consisting of several tasks.

This paper is a extended version of [2].

1. While our previous work only considers delegations of single task, we here generalize our formal framework to consider any bilateral reallocation (delegation or swap of several tasks) in order to reduce not only the mean flowtime but also the rescheduling time.
2. We here redefine the acceptability criterion of the bilateral reallocations by the agents. Previously, this criterion was based on the local flowtime, i.e. the flowtime restricted to the two contractors. Since this criterion does not guarantee the termination of the multi-agent reallocation algorithm, it was combined with the *makespan*, the maximum completion of all the jobs. In this paper, a bilateral reallocation is acceptable for an agent if, according to its beliefs, it reduces the global flowtime. This acceptability criterion is sufficient to guarantee the convergence of the reallocation process. Moreover, it allows to reduce the rescheduling time and the mean flowtime reached by our strategy.
3. We quantitatively compare the performance of our strategy with the performance of a DCOP resolution method.

3 Multi-agent situated task allocation

In this section, we formalize the multi-agent situated task allocation problem with concurrent jobs.

We consider distributed jobs, each job being a set of independent, non divisible tasks without precedence order. Tasks are non preemptive, and the execution of each task requires resources which are distributed across different nodes. These resources are transferable and non consumable.

Definition 1 (Distributed system). *A system is a triple* $\mathcal{D} = \langle \mathcal{N}, \mathcal{E}, \mathcal{R} \rangle$ *where:*

- $\mathcal{N} = \{\nu_1, \ldots, \nu_m\}$ *is a set of m nodes;*
- $\mathcal{E}$ *is an acquaintance relation, i.e. a binary and symmetric relation over* $\mathcal{N}$*;*
- $\mathcal{R} = \{\rho_1, \ldots, \rho_k\}$ *is a set of k resources, each resource* ρ_i *having a size* $|\rho_i|$*. The locations of the resources, which are possibly replicated, are determined by the function:*

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

For simplicity, we assume exactly one agent per node (the set of agents is $\mathcal{N}$), and any agent can access any resource.

Running a job (without a deadline) consists in a set of independent tasks which require resources to produce an outcome.

Definition 2 (Job/Task). *Let $\mathcal{D}$ be a distributed system and Res be the space of outcomes. We consider the set of ℓ jobs $\mathcal{J} = \{J_1, \ldots, J_\ell\}$. Each job J_i is a set of k_i tasks $J_i = \{\tau_1, \ldots, \tau_{k_i}\}$ where each task τ is a function which links a set of resources to an outcome: $\tau : 2^{\mathcal{R}} \mapsto$ Res.*

It is worth noticing that we consider in this paper a set of jobs having the same release date. $\mathcal{T} = \bigcup_{1 \leq i \leq \ell} \mathrm{J}_i$ denotes the set of the n tasks of $\mathcal{J}$ and $\mathcal{R}_\tau \subseteq \mathcal{R}$ is the set of the resources required for the task τ. The job containing the task τ is written $\mathrm{job}(\tau)$.

Each task has a cost for a node, which is its estimated execution time by this node. As the fetching time of resources is supposed to be significant, the cost function must verify that the task τ is cheaper for ν_i than for ν_j if the required resources are "more local" to ν_i than to ν_j:

Property 1 (Cost). Let $\mathcal{D}$ a distributed system and $\mathcal{T}$ a set of tasks. The cost function $c : \mathcal{T} \times \mathcal{N} \mapsto \mathbb{R}^*_+$ is such that:

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

The cost function can be extended to a set of tasks:

$$\forall \mathrm{T} \subseteq \mathcal{T},\ \mathrm{c}(\mathrm{T}, \nu_i) = \Sigma_{\tau \in \mathrm{T}} \mathrm{c}(\tau, \nu_i) \tag{3}$$

Note that in practice, it is difficult to design this function with a good estimation of the runtime. However the adaptability of our multi-agent system compensates for a poor estimation of the cost function. It is one of the benefits of our approach.

A multi-agent situated task allocation problem with concurrent jobs consists in assigning several jobs to some nodes according to their costs.

Definition 3 (MASTA+). *A multi-agent situated task allocation problem with concurrent jobs is a quadruple MASTA+ $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ where:*

- *$\mathcal{D}$ is a distributed system with m nodes ;*
- *$\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is a set of n tasks ;*
- *$\mathcal{J} = \{J_1, \ldots, J_\ell\}$ is a partition of $\mathcal{T}$ into ℓ jobs;*
- *$c : \mathcal{T} \times \mathcal{N} \mapsto \mathbb{R}^*_+$ is the cost function.*

A task allocation is an assignation of sorted bundles to different nodes.

Definition 4 (Allocation). *Let MASTA+ be a task allocation problem. An allocation $\overrightarrow{A}$ is a vector of m sorted task bundles $((B_1, \prec_1), \ldots, (B_m, \prec_m))$. Each bundle $(B_i, \prec_i)$ is the set of tasks ($B_i \subseteq \mathcal{T}$) assigned to the node ν_i associated with a scheduling order, i.e. a strict and total order ($\prec_i \subseteq \mathcal{T} \times \mathcal{T}$) such that $\tau_j \prec_i \tau_k$ means that if $\tau_j, \tau_k \in B_i$ then τ_j is performed before τ_k by ν_i.*

The allocation $\overrightarrow{A}$ *is such that:*

$$\forall \tau \in \mathcal{T},\ \exists \nu_i \in \mathcal{N},\ \tau \in B_i \tag{4}$$

$$\forall \nu_i \in \mathcal{N}, \forall \nu_j \in \mathcal{N} \setminus \{\nu_i\},\ B_i \cap B_j = \emptyset \tag{5}$$

The set $\mathcal{T}$ is partionned by an allocation: all the tasks are assigned (Eq. 4) and each task is assigned to a single node (Eq. 5). To simplify, we use the following notations:

- $\overrightarrow{\mathrm{B}}_i = (\mathrm{B}_i, \prec_i)$, the sorted bundle of ν_i;
- $\min_{\prec_i} \mathrm{B}_i$, the next task to perform by ν_i:
- $\mathrm{jobs}(\mathrm{B}_i)$, the set of jobs assigned to ν_i, i.e. such that at least one task is in B_i;
- $\nu(\tau, \overrightarrow{\mathrm{A}})$, the node whose bundle contains τ in $\overrightarrow{\mathrm{A}}$;
- $\mathrm{w}_i(\overrightarrow{\mathrm{A}}) = \mathrm{c}(\mathrm{B}_i, \nu_i) = \Sigma_{\tau \in \mathrm{B}_i} \mathrm{c}(\tau, \nu_i)$, the workload of ν_i for $\overrightarrow{\mathrm{A}}$.

We assume that nodes are never idle, so the completion time of a task is its delay before the task is started, plus its estimated execution time:

$$\mathrm{C}_\tau(\overrightarrow{\mathrm{A}}) = \mathrm{t}(\tau, \nu(\tau, \overrightarrow{\mathrm{A}})) + \mathrm{c}(\tau, \nu(\tau, \overrightarrow{\mathrm{A}})) \tag{6}$$
$$\text{with } \mathrm{t}(\tau, \nu_i) = \Sigma_{\tau' \in \mathrm{B}_i | \tau' \prec_i \tau} \mathrm{c}(\tau', \nu_i)$$

Unlike the cost, the delay (so the completion time) depends on the scheduling order over the bundle.

The quality of an allocation is measured by the mean flowtime for all the jobs, where the flowtime of one job is its completion time. The makespan is the time necessary to perform all the jobs. Then, the makespan is the maximum completion time of the jobs and also the maximum workload of the nodes.

Definition 5 (Flowtime/Makespan). *Let MASTA+ a task allocation problem and* $\overrightarrow{A}$ *an allocation. We define:*

- *the completion time of* $J \in \mathcal{J}$ *for* $\overrightarrow{A}$*,*

$$C_J(\overrightarrow{A}) = \max_{\tau \in J} \{C_\tau(\overrightarrow{A})\} \tag{7}$$

- *the (mean) flowtime of* $\mathcal{J}$ *for* $\overrightarrow{A}$*,*

$$C_{mean}(\overrightarrow{A}) = \frac{1}{\ell} C(\overrightarrow{A}) \text{ with } C(\overrightarrow{A}) = \Sigma_{J \in \mathcal{J}} C_J(\overrightarrow{A}) \tag{8}$$

- *the makespan of* $\mathcal{J}$ *for* $\overrightarrow{A}$*,*

$$C_{max}(\overrightarrow{A}) = \max_{\nu_i \in \mathcal{N}} \{w_i(\overrightarrow{A})\} \tag{9}$$

- *the local availability ratio of* $\overrightarrow{A}$*,*

$$L(\overrightarrow{A}) = \Sigma_{\tau \in \mathcal{T}} \frac{\Sigma_{\rho \in \mathcal{R}_\tau,\ \nu(\tau, \overrightarrow{A}) \in l(\rho)} |\rho|}{\Sigma_{\rho \in \mathcal{R}_\tau} |\rho|} \tag{10}$$

Unlike the makespan, the flowtime depends on the scheduling order. The local availability ratio of $\overrightarrow{\mathrm{A}}$ measures the proportion of locally processed resources (Eq. 10).

Example 1 (MASTA+). From the distributed system $\mathcal{D} = \langle \mathcal{N}, \mathcal{E}, \mathcal{R} \rangle$ with $\mathcal{N} = \{\nu_1, \nu_2, \nu_3\}$, $\mathcal{E} = \{(\nu_1, \nu_2), (\nu_1, \nu_3), (\nu_2, \nu_3)\}$ and $\mathcal{R} = \{\rho_1, \ldots, \rho_9\}$ where resources are replicated on 2 nodes (cf. Fig. 1a), we consider MASTA+ $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ with $\mathcal{T} = \{\tau_1, \ldots, \tau_9\}$ where each task τ_i needs resource ρ_i, $\mathcal{J} = \{\mathrm{J}_1, \mathrm{J}_2, \mathrm{J}_3\}$ such that $\mathrm{J}_1 = \{\tau_1, \tau_2, \tau_3\}$, $\mathrm{J}_2 = \{\tau_4, \tau_5, \tau_6\}$ and $\mathrm{J}_3 = \{\tau_7, \tau_8, \tau_9\}$ and c is the cost function given in Table 2. We assume the cost of a task is proportional to the resources size, and two times greater if the resource is distant. We consider here the allocation $\overrightarrow{\mathrm{A}}$ (see Figure 1b) with $\overrightarrow{\mathrm{B}}_1 = (\tau_5, \tau_8, \tau_3, \tau_2)$, $\overrightarrow{\mathrm{B}}_2 = (\tau_4, \tau_9)$ and $\overrightarrow{\mathrm{B}}_3 = (\tau_7, \tau_1, \tau_6)$. The makespan and the flowtime are $\mathrm{C}_{max}(\overrightarrow{\mathrm{A}}) = 12$ and $\mathrm{C}(\overrightarrow{\mathrm{A}}) = 8 + 12 + 12 = 32$.

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

Table 2: Cost function

To conclude this section, due to the locality of resources, a task has not the same cost for every nodes. In this paper, our objective is to minimize the mean flowtime, for a set of concurrent jobs composed of many tasks.

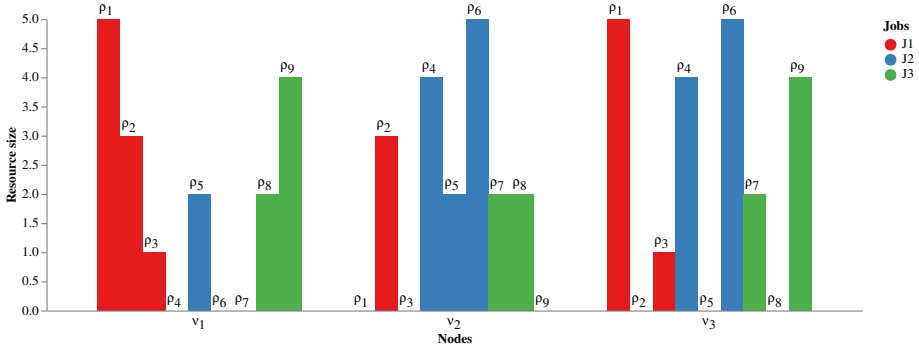
4 Consumption and reallocation

We describe in this section the operations of consumption and reallocation as well as the negotation protocol.

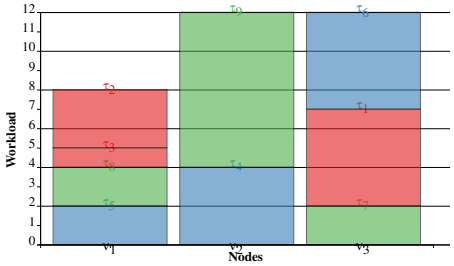
A task **consumption** is the removal by a node of a task from its bundle in order to process it. This operation modifies not only the current allocation but also the underlying MASTA+ problem since the consumed task is no longer present. The consumption strategy adopted by an agent specifies the tasks scheduling for the node it is in charge of. Since we aim at minimizing the mean flowtime of the jobs, we consider here a job-oriented strategy which sorts first jobs and then the tasks inside the same job (the tasks of a same job are consecutive in the bundle). More precisely, the less expensive jobs are prior on the most expensive ones in order to minimize locally the completion time of the jobs. Thereafter, $\mathrm{J}_1 \blacktriangleleft_i \mathrm{J}_2$ means that the tasks in J_1 are prior to the tasks in J_2. $\tau_1 \lhd_i \tau_2$ means that the task τ_1 is prior to the task τ_2. Formally,

$$\begin{aligned} &\forall \tau_j, \tau_k \in \mathrm{B}_i \; \tau_j \prec_i \tau_k \Leftrightarrow \\ &\mathrm{job}(\tau_j) \blacktriangleleft_i \mathrm{job}(\tau_k) \vee (\mathrm{job}(\tau_j) = \mathrm{job}(\tau_k) \wedge \tau_j \lhd_i \tau_k) \end{aligned} \tag{11}$$

The addition/removal of a list of tasks T in the bundle $\overrightarrow{\mathrm{B}}_i$ of the node ν_i may modify the tasks execution order since these operations imply a rescheduling of the bundle:



(a) Resource distribution



(b) Task allocation

Fig. 1: Resource distribution and task allocation for example 1

- $\overrightarrow{\mathrm{B}_i \oplus \mathrm{T}}$ denotes the bundle containing the set of tasks $\mathrm{B}_i \cup \mathrm{T}$ sorted with $\prec_i$;
- $\overrightarrow{\mathrm{B}_i \ominus \mathrm{T}}$ denotes the bundle containing $\mathrm{B}_i \setminus \mathrm{T}$ sorted with $\prec_i$.
- $\overrightarrow{\mathrm{B}_i \ominus \mathrm{T}_1 \oplus \mathrm{T}_2}$ denotes the bundle containing $\mathrm{B}_i \setminus \mathrm{T}_1 \cup \mathrm{T}_2$ sorted with $\prec_i$.

A **bilateral reallocation** is an operation which modifies the current allocation by exchanging one or several tasks between two agents.

Definition 6 (Bilateral reallocation). *Let MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *be an allocation problem and* $\overrightarrow{A} = (\overrightarrow{B}_1, \ldots, \overrightarrow{B}_m)$ *an allocation. The bilateral reallocation of the non-empty list of tasks* T_1 *assigned to the proposer* ν_i *in exchange for the list of tasks* T_2 *assigned to the responder* ν_j *in* $\overrightarrow{A}$ *(*$\mathrm{T}_1 \subseteq B_i$ *and* $\mathrm{T}_2 \subseteq B_j$*) leads to the allocation* $\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A})$ *with m bundles* $\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{B}_k)$ *such that:*

$$\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{B}_k) = \begin{cases} \overrightarrow{B_i \ominus \mathrm{T}_1 \oplus \mathrm{T}_2} & \text{if } k = i, \\ \overrightarrow{B_j \ominus \mathrm{T}_2 \oplus \mathrm{T}_1} & \text{if } k = j, \\ \overrightarrow{B}_k & \text{otherwise} \end{cases} \tag{12}$$

We distinguish two cases:

- a **swap** where the two lists of tasks are non-empty ($\mathrm{T}_1 \neq \emptyset \wedge \mathrm{T}_2 \neq \emptyset$), denoted $\sigma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{\mathrm{A}})$;
- a **delegation** where an agent gives a part of its tasks to one of its peers without counterpart ($\mathrm{T}_2 = \emptyset$), denoted $\delta(\mathrm{T}_1, \nu_i, \nu_j, \overrightarrow{\mathrm{A}})$. If $|\mathrm{T}_1| = 1$, this is an **unary delegation**. Otherwise, this is an ***n*-ary delegation**.

We will see later that the bilateral reallocation of lists of tasks rather than sets allows to specify the order in which the tasks should be evaluated to validate the interest of all or part of the transaction.

In order to improve an allocation, we introduce the notion of socially rational bilateral reallocation which verifies if a reallocation reduces the global flowtime, i.e. the completion time of the jobs for all nodes.

Definition 7 (Socially rational bilateral reallocation). *Let MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *be an allocation problem,* $\overrightarrow{A}$ *an allocation. The bilateral reallocation* $\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A})$ *is socially rational with respect to the flowtime if and only if the global flowtime decreases,*

$$C(\gamma(\mathrm{T}_1, \mathrm{T}_2, \nu_i, \nu_j, \overrightarrow{A})) < C(\overrightarrow{A}) \tag{13}$$

An allocation is **stable** if there is no socially rational bilateral delegation.

Contrary to [2], we do not consider as socially rational the reallocations reducing the local flowtime (the completion time of jobs restricted to the nodes implied in the reallocation) which does not guarantee the convergence of the reallocation process, nor even the reallocations reducing the local flowtime and the makespan (the maximum workload of the agents). The reduction of the global flowtime guarantees the termination of the process. Hereafter, when it comes to flowtime, it will be, unless specified, the global flowtime (denoted $\mathrm{C}(\overrightarrow{\mathrm{A}})$ defined in Eq. 8)

Property 2 (Termination). Let MASTA+ $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ be an allocation problem and $\overrightarrow{\mathrm{A}}$ a non-stable allocation with respect to the flowtime. There exists a finite path of socially rational bilateral reallocations with respect to the flowtime which leads to a stable allocation for this criterion.

Proof. Let MASTA+ $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ be an allocation problem and $\overrightarrow{\mathrm{A}}$ a non-stable allocation with respect to the flowtime. Let γ be a socially rational reallocation which leads to the allocation $\overrightarrow{\mathrm{A}}'$ from $\overrightarrow{\mathrm{A}}$. Since γ is socially rational with respect to the flowtime, the flowtime strictly decreases. Formally, $\Sigma_{\mathrm{J} \in \mathcal{J}} C_J(\overrightarrow{\mathrm{A}}') < \Sigma_{\mathrm{J} \in \mathcal{J}} \mathrm{C_J}(\overrightarrow{\mathrm{A}})$. Since there is a finite number of allocations and $\Sigma_{\mathrm{J} \in \mathcal{J}} \mathrm{C_J}(\overrightarrow{\mathrm{A}})$ strictly decreases at each step, there can only be a finite number of such allocations.

For tasks reallocation, the agents are involved in multiple bilateral single-round negotiation. Each negotiation is based on the alternating offers protocol [18] and includes three decision steps : (a) the offer strategy of the proposer which selects a delegation, i.e. a list of tasks in its bundle and a responder, (b) the counter-offer strategy which allows the responder to determine wether it declines, accepts or makes a counter-offer to the delegation, and (c) the eventual reallocation is confirmed or withdrawn by the proposer according to the consumptions that happen concurrently (cf. figure 2).

Example 2 (Consumption and reallocation). Let us consider the problem MASTA+ from the example 1 and the allocation $\overrightarrow{\mathrm{A}}$ in Figure 1b. According to the consumption strategy adopted by the agents, each bundle is sorted. The less expensive jobs are prior (e.g. $\mathrm{J}_3 \blacktriangleleft_3 \mathrm{J}_1 \blacktriangleleft_3 \mathrm{J}_2$). In case of a tie, the natural order over the identifiers ensures a strict and total order (e.g. $\mathrm{J}_2 \blacktriangleleft_1 \mathrm{J}_3$). The tasks among a same job are sorted by increasing cost ($\tau_3 \lhd_1 \tau_2$). The delegation of the task τ_9 by the node ν_2 to the node ν_1 leads to the allocation $\overrightarrow{\mathrm{A}}' = \delta([\tau_9], \nu_2, \nu_1, \overrightarrow{\mathrm{A}})$. This delegation (cf. Figure 3a) is socially rational since it decreases the flowtime from 32 to 31. The swap of $\tau_9 \in \mathrm{B}_2$ and $\tau_5 \in \mathrm{B}_1$ between the nodes ν_2 and ν_1 leads to the allocation $\overrightarrow{\mathrm{A}}'' = \sigma([\tau_9], [\tau_5], \nu_2, \nu_1, \overrightarrow{\mathrm{A}})$. This swap (cf. Figure 3b) decreases the flowtime from 32 to 29.

5 Negotiation strategy

We describe in this section the different parts of the negotiation strategy and we sketch the agent behaviour in the negotiation process.

5.1 Peer modelling

The peer modelling is built upon exchanged information through messages between the agents. Before the negotiation process and between each bilateral reallocation it is implied in, the agent ν_i informs its peers about the cost of each job J for it ($\mathrm{c}(\mathrm{J}, \nu_i)$). Since the number of jobs is negligible compared to the number of tasks, the messages size is insignificant compared to the bundle descriptions. The modelling for the target ν_k by the subject ν_i is based on:

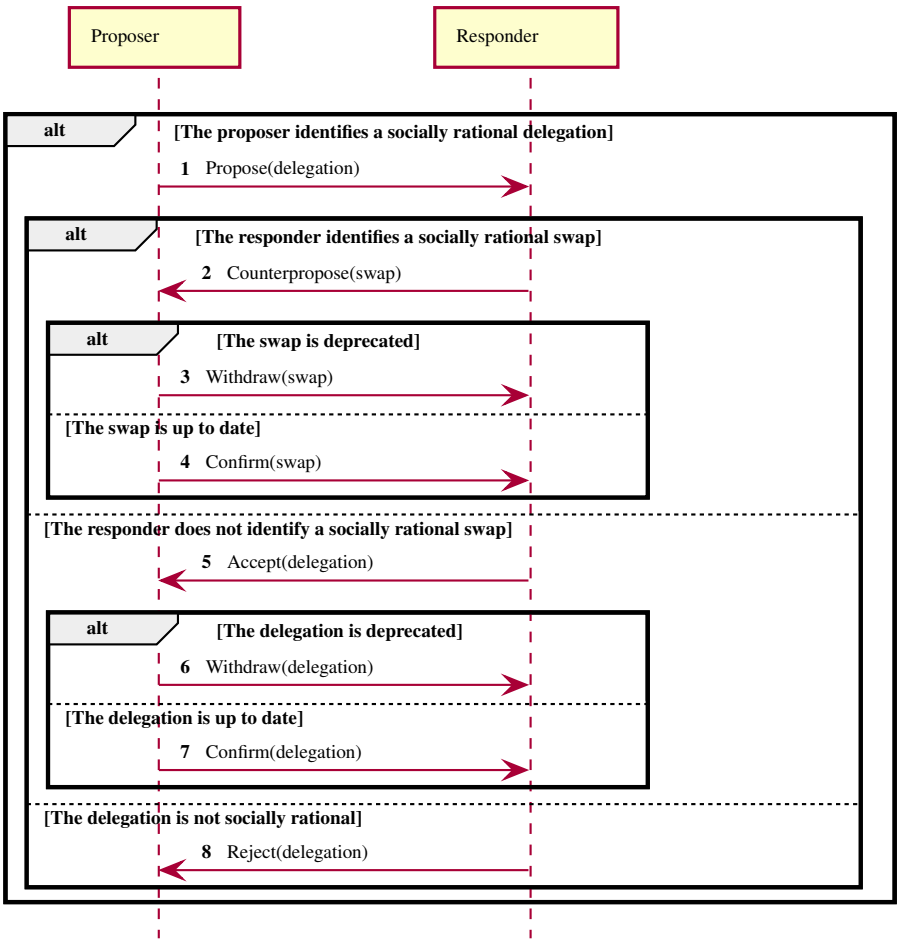
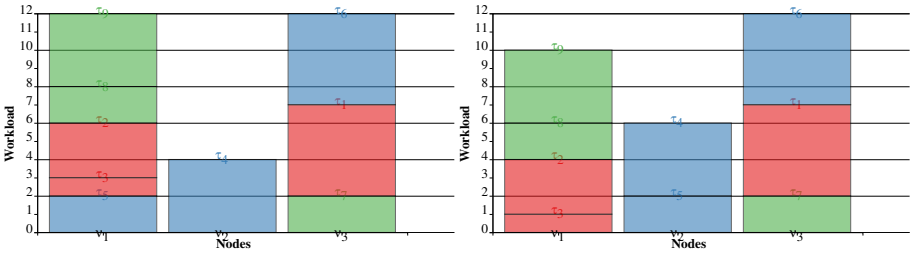


Fig. 2: Bilateral negotiation protocol between a proposer and a responder.



(a) Allocation after the delegation of τ_9 from ν_2 to ν_1

(b) Allocation after the swap of τ_9 and τ_5 between ν_2 and ν_1

Fig. 3: Allocations of the first example resulting from bilateral reallocations

1. the belief base of the subject, possibly partial or obsolete, which contains the beliefs about the costs of the jobs for ν_k ($c^i(J,\nu_k)$, $\forall J \in \mathcal{J}$) and so the beliefs about the workload of ν_k ($w^i_k(\overrightarrow{A}) = \Sigma_{J\in\mathcal{J}} c^i(J,\nu_k)$);
2. the consumption strategy of the target assumed by the subject, written $(\mathcal{J}, \blacktriangleleft^i_k)$.

The subject can then deduce:

- the completion time ($C_J{}^i(\overrightarrow{B}_k)$) of the job J for a target k, possibly itself ($\nu_k = \nu_i$), after the addition ($C_J{}^i(\overrightarrow{B_k \oplus T})$), the removal ($C_J{}^i(\overrightarrow{B_k \ominus T})$) and the replacing of tasks ($C_J{}^i(\overrightarrow{B_k \ominus T_1 \oplus T_2})$);
- the completion time of a job J for the allocation,

$$C_J{}^i(\overrightarrow{A}) = \max_{\nu_k \in \mathcal{N}} C_J{}^i(\overrightarrow{B}_k) \text{ où } C_J{}^i(\overrightarrow{B}_i) = C_J(\overrightarrow{B}_i) \tag{14}$$

- the **bottleneck node** for each job J, denoted $\nu^i_{max}(\overrightarrow{A}, J)$, i.e. the node ν_k for which the completion time of this job is the maximum completion time in the allocation,

$$C_J{}^i(\overrightarrow{B}_k) = C_J{}^i(\overrightarrow{A}) \tag{15}$$

- the flowtime of the allocation $\overrightarrow{A}$

$$C^{\,i}(\overrightarrow{A}) = \Sigma_{J\in\mathcal{J}} C_J{}^i(\overrightarrow{A}) \tag{16}$$

5.2 Acceptability rule

The **acceptability rule** is a local decision made by an agent which is implied in a bilateral reallocation. This rule, which is based on the agent knowledge and the peer modelling, decides to accept or decline a reallocation.

Definition 8 (Acceptability). *Let MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *be a problem and* $\overrightarrow{A}$ *an allocation. The bilateral reallocation* $\gamma(T_1, T_2, \nu_i, \nu_j, \overrightarrow{A})$ *is acceptable by the agent* $\nu_k \in \mathcal{N}$ *with respect to the flowtime if an only if the agent believes that the flowtime decreases,*

$$\Sigma_{J\in\mathcal{J}} \max_{\forall \nu_o \in \mathcal{N}\setminus\{\nu_i,\nu_j\}} (C^k_J(\overrightarrow{B_i \ominus T_1 \oplus T_2}), C^k_J(\overrightarrow{B_j \ominus T_2 \oplus T_1}), C^k_J(B_o)) < C^{\,k}(\overrightarrow{A}) \tag{17}$$

The acceptability with respect to the flowtime is based on the beliefs about the completion time of the jobs for all the nodes before and after the reallocation (Eq. 17).

We propose in this paper a process where the agents trigger concurrent bilateral negotiations leading to socially rational reallocations.

5.3 Offer strategy

The **offer strategy** of an agent, which is based on its knowledge, its beliefs and its peer modelling, identifies a delegation in three steps. An agent ν_i selects an offer bundle, i.e.

a list of tasks to delegate to a responder in a set $\mathcal{N}'$ in order to reduce the completion time of a job in a set $\mathcal{J}'$ for which it is a bottleneck. Initially, $\mathcal{J}' = \mathcal{J}$, $\mathcal{N}' = \mathcal{N}$.

1. Job selection. In order to reduce not only the completion time of a job but also the completion time of the next jobs in its bundle, our heuristic selects the most prior job J_* for which it is a bottleneck,

$$J_* = \min_{\blacktriangleleft_i}\{J \in \text{jobs}(B_i) \cap \mathcal{J}' \mid \nu^i_{\max}(\overrightarrow{A}, J = \nu_i)\} \tag{18}$$

2. Responder selection. The jobs of a responder that are impacted by the delegation are those after J_* according to $\blacktriangleleft^i_j$. Not to increase the completion time of these jobs, our heuristic selects a responder ν_* for whom the sum of the differences between the completion time for the allocation and the completion time for the agent is the greatest one,

$$\nu_* = \mathit{random}\{\operatorname*{argmax}_{\nu_j \in \mathcal{N}'} \Sigma_{J_* \blacktriangleleft^i_j J}(C_J{}^i(\overrightarrow{A}) - C_J{}^i(\overrightarrow{B}_j))\} \tag{19}$$

where *random* is a random choice function which selects a node from any set of nodes.

3. Offer bundle selection. To determine the offer bundle, we distinguish a strategy which selects a single task as in [2] and a strategy which selects several tasks.

3.a. Unary delegation selection. In order to reduce the completion time of J_*, the proposer selects a distant task, i.e. a task whose delegation will reduce its cost. Our heuristic selects the task in the job J_* or in the prior jobs in $\overrightarrow{B}_i$ with the best payoff in terms of cost. In case of a tie, the prior task is chosen,

$$\forall \mathcal{T}' \subseteq \mathcal{T}, \tau_* = \min_{\lhd_i}\{\operatorname*{argmax}_{\tau \in \mathcal{T}' \cap B_i \cap \{J \mid J = J_* \vee (J \blacktriangleleft_i J_*)\}} c(\tau, \nu_i) - c(\tau, \nu_*)\}$$

The delegation $\delta([\tau_*], \nu_i, \nu_*, \overrightarrow{A})$ is triggered if it is acceptable for the proposer (cf Def. 8).

3.b. *N*-ary delegation selection. The proposer iteratively builds an offer bundle T_*. This bundle is a stack of tasks that will be evaluated by the acceptability strategy of the responder in order to accept all or part of this bundle by unstacking it (cf. Section 5.4). As illustrated in Algorithm 1, our heuristic considers the tasks in J_* or in the prior jobs in $\overrightarrow{B}_i$ (line 2). The proposer ν_i selects in priority the distant tasks, i.e. the tasks whose delegation will reduce at most the processing time (lines 3 and 6). According to this ratchet algorithm, the flowtime strictly decreases during the building of the offer bundle (line 8). Moreover, the algorithm stops as soon as a task does not improve the flowtime. If the offer bundle T_* is non-empty, the delegation $\delta(T_*, \nu_i, \nu_*, \overrightarrow{A})$, which is acceptable for the initiator, is triggered.

Whatever the offer bundle selection strategy is (3.a or 3.b), if no delegation is triggered, the offer strategy returns to step #2 to choose another responder ($\mathcal{N}' \leftarrow \mathcal{N}' \setminus \{\nu_*\}$). Otherwise, the offer strategy returns to step #1 to choose another job ($\mathcal{J}' \leftarrow \mathcal{J}' \setminus \{J_*\}$). In case of failure, no delegation is proposed and the agent goes into pause state until its belief base is updated and a new opportunity (i.e. a delegation) is found.

Algorithm 1: Building of the offer bundle by the proposer ν_i

Data: J_* the job selected in step #1;
ν_* the responder selected in step #2;

$T_* \leftarrow$ `empty_stack` ;
$T = \{\tau \mid \text{job}(\tau) = J_* \vee (\text{job}(\tau) \blacktriangleleft_i J_*)\}$;
$T' \leftarrow (\ldots,\tau^k,\ldots,\tau^l,\ldots) \mid \tau^i \in T \wedge (k < l \Leftrightarrow c(\tau^k,\nu_i) - c(\tau^k,\nu_*) > c(\tau^l,\nu_i) - c(\tau^l,\nu_*))$
`/* the list of tasks by decreasing payoff */`
$bestFlowtime = C^i(\overrightarrow{A})$;
while $T' \neq \emptyset$ **do**
  $\tau_* \leftarrow$ `head`(T') ;
  $T' \leftarrow$ `tail`(T') ;
  if $C^i(\delta(T_* \cup \{\tau_*\}, \nu_i, \nu_*, \overrightarrow{A})) < bestFlowtime$ **then**
    T_*.`push`(τ_*) ;
    $bestFlowtime \leftarrow C^i(\delta(T_*, \nu_i, \nu_*, \overrightarrow{A}))$;
  end
  else
    Return T_* ;
  end
end
Return T_* ;

Algorithm 2: Selection by the responder ν_* of a sub-bundle among the received offer bundle

Data: T_* the received offer bundle

$T_{acc} \leftarrow T_*$;
while $\delta(T_{acc}, \nu_i, \nu_*, \overrightarrow{A})$ *is not acceptable for the agent* ν_* **do**
  T_{acc}.`pop` ;
end
Return T_{acc} ;

5.4 Acceptation strategy

According to the **acceptation strategy**, the responder accepts a delegation which is acceptable for it. Otherwise, it unstacks one by one the tasks in the offer bundle T_* (cf. Algo. 2) for possibly accepting a part of it. When the sub-bundle T_{acc} is empty, the responder declines the offer.

5.5 Agent behaviour

In our approach, a reallocation is the outcome of the negotiation process between agents adopting the same behaviour. The agent behaviour is specified in [3] by a deterministic finite state automaton [1]. An agent executes its behaviour according to its knowledge and its beliefs. In order to avoid deadlock, the proposals are associated with deadlines. The agent's belief base is updated by the reception of messages. None proposal is sent if the agent believes that the allocation is stable.

Example 3 (Negotiation strategy). Let us consider the MASTA+ problem from Example 1 and the initial allocation $\overrightarrow{A}$ (cf. Figure 4a) such that $\overrightarrow{B}_1 = (\tau_5, \tau_1)$, $\overrightarrow{B}_2 = (\tau_3, \tau_2, \tau_7, \tau_8, \tau_9)$ and $\overrightarrow{B}_3 = (\tau_4, \tau_6)$. The flowtime is $C(\overrightarrow{A}) = 7 + 9 + 17 = 33$. We consider that the belief bases are up-to-date. The offer strategy of the agent ν_2 selects a delegation as follows:

1. it selects the most prior job for which it is a bottleneck (cf. Eq. 18), $J_* = J_3$;
2. it selects an agent which is the least bottleneck for the job J_3 (cf. Eq. 19). As neither ν_1 nor ν_3 have tasks from J_3, the agent ν_2 randomly chooses, $\nu_* = \nu_1$;
3. The algorithm 1 allows the agent ν_2 to select its offer bundle :
   (a) the candidate tasks, i.e. the tasks in J_3 or the previous ones in its bundle, are sorted by decreasing payoff, $T' = [\tau_9, \tau_3, \tau_2, \tau_8, \tau_7]$;
   (b) the delegation of the task τ_9 improves the flowtime (cf. Figure 4b),

$$C^2(\delta([\tau_9], \nu_2, \nu_1, \overrightarrow{A})) = 11 + 9 + 9 = 29 < 33 \tag{20}$$

   The task τ_9 is added to the offer bundle, $T_* = [\tau_9]$,
   (c) the delegation of the tasks τ_3 and τ_9 improves the flowtime (cf. Figure 4c),

$$C^2(\delta([\tau_9, \tau_3], \nu_2, \nu_1, \overrightarrow{A})) = 12 + 9 + 7 = 28 < 29 \tag{21}$$

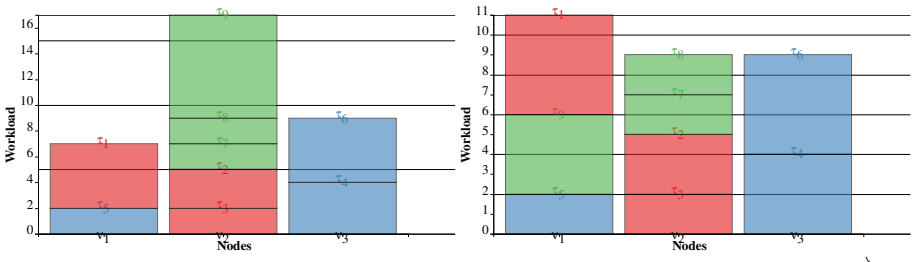
   The task τ_3 is added to the offer bundle, $T_* = [\tau_9, \tau_3]$,
   (d) The delegation of the tasks τ_2, τ_3 and τ_9 does not improve the flowtime (cf. Figure 4c),

$$C^2(\delta([\tau_9, \tau_3, \tau_2], \nu_2, \nu_1, \overrightarrow{A})) = 15 + 9 + 6 = 30 > 28 \tag{22}$$

   The selected offer bundle is $T_* = [\tau_9, \tau_3]$.

In summary, the agent ν_2 proposes the delegation $\delta([\tau_9, \tau_3], \nu_2, \nu_1, \overrightarrow{A})$ to the agent ν_1.

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



(a) Initial allocation

(b) After the delegation $\delta([\tau_9], \nu_2, \nu_1, \overrightarrow{A})$

(c) After the delegation $\delta([\tau_9, \tau_3], \nu_2, \nu_1, \overrightarrow{A})$

(d) After the delegation $\delta([\tau_9, \tau_3, \tau_2], \nu_2, \nu_1, \overrightarrow{A})$

Fig. 4: Allocations from Example 3

6 Results and discussion

After having presented the experimental context, we empirically compare our approach with a classic heuristic and with a distributed constraint optimization (DCOP) resolution method. Moreover, we consider our new acceptability criterion and *n*-ary delegation.

6.1 Context of experiments

The practical application we consider is the distributed deployment of the MapReduce design pattern in order to process large datasets on a cluster, as with Spark [24]. We focus here on the *Reduce* stage of MapReduce jobs. This can be formalized by a MASTA+ problem where several jobs are concurrently submitted and the cost function is s.t.:

$$\begin{aligned} c_i(\tau, \nu_j) &= \sum_{\rho_j \in \mathcal{R}_\tau} c_i(\rho_j, \nu_j) \\ \text{with } c_i(\rho_j, \nu_i) &= \begin{cases} |\rho_j| \text{ if } \nu_i \in l(\rho_j) \\ \kappa \times |\rho_j| \text{ else} \end{cases} \end{aligned} \tag{23}$$

where we empirically calibrate $\kappa = 2$ as a realistic value to capture the overhead due to remote resource fetching.

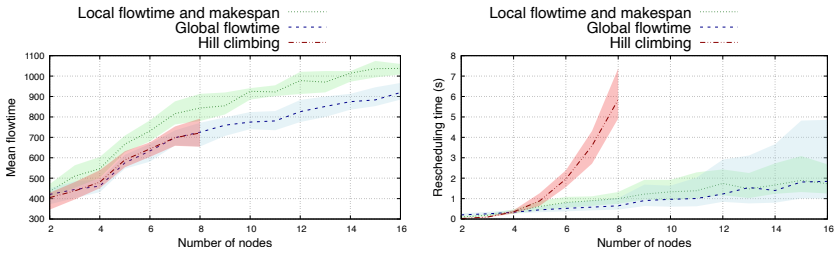
Our prototype [3] is implemented with the programming language Scala and Akka [16] for highly concurrent, distributed, and resilient message-driven applications. We assume

that: (a) the message transmission delay is arbitrary but not negligible, (b) the message order per sender-receiver pair is preserved, and (c) the delivery of messages is guaranteed. Experiments have been conducted on a blade with 20 CPUs and 512Go RAM.

This work is a first step for evaluating our stategies. Indeed we compare the reallocation process, i.e. a MASTA+ problem solving, without considering the iterations induced by task consumptions, even if the task consumption strategy is required to sort the agent's task bundle. We consider MASTA+ problem instance such that $m \in [2;16]$ nodes/agents, $\ell = 4$ jobs, $n = 3 \times \ell \times m$ tasks, with one resource per task. Each resource ρ_i is replicated 3 times and $|\rho_i| \in [0;100]$. We generate 10 MASTA+ problem instances, and for each we randomly generate 10 initial allocations. We assess the medians and the standard deviations of three metrics: (1) the mean flowtime (Eq. 8), (2) the local availability ratio (Eq. 10), and (3) the rescheduling time.

6.2 Classical heuristic and acceptability criterion

The hypothesis we want to test are: (1) the flowtime reached by our strategy is close to the one reached by the classical approach and (2) the decentralization significantly reduces the scheduling time. Moreover, unlike our previous work [2] where we used the local flowtime and the makespan in our acceptability criterion, here, we only consider the global flowtime which is sufficient to ensure the negotiation process convergence. We also want to verify that the acceptability criterion allows to significantly improve the quality of the outcome.



(a) Flowtime (b) Scheduling time

Fig. 5: The flowtime and the rescheduling time of our strategy, the strategy in [2] and an hill climbing algorithm.

Figures 5a and 5b respectively compare the flowtime and rescheduling time of our unary delegation strategy with the strategy presented in [2] and an hill climbing algorithm. These three algorithms start with the same random initial allocation. At each step, the hill climbing algorithm selects among all the possible delegations, the one which minimizes the flowtime.

In Figure 5a, we observe that, while the quality of the solution reached by the strategy proposed in [2] is slightly lower than the one reached with the hill climbing algorithm, our strategy now reaches similar solutions. This is due to the fact that a socially

rational reallocation according to the global flowtime can only decrease the flowtime, while it is not the case when the local flowtime is used. Moreover, since the acceptability criterion is no more based on the makespan, the number of possible delegations, which can improve the flowtime, increases.

Figure 5b shows that the rescheduling time of our new strategy remains approximately the same as the former one. Then, it is much better than the rescheduling time of the hill climbing algorithm which exponentially grows with the number of nodes. Thus, the acceptability criterion we proposed in this article significantly improves the flowtime with a similar rescheduling time. It is worth noticing that the hill climbing algorithm has been used with small MASTA+ instances due to its prohibitive scheduling time. One can expect to obtain a greater rescheduling time with a local search method, such as simulated annealing, with no guarantee to have a more qualitative outcome. As a result, even if the number of agents is small, the gain realized on the flowtime by the hill climbing algorithm will be penalized and cancelled by the overhead of its scheduling time. This overhead penalized the time-extented assignment in a distributed system which should be adaptive to disruptive phenomena (task consumption, job release, slowing down nodes).

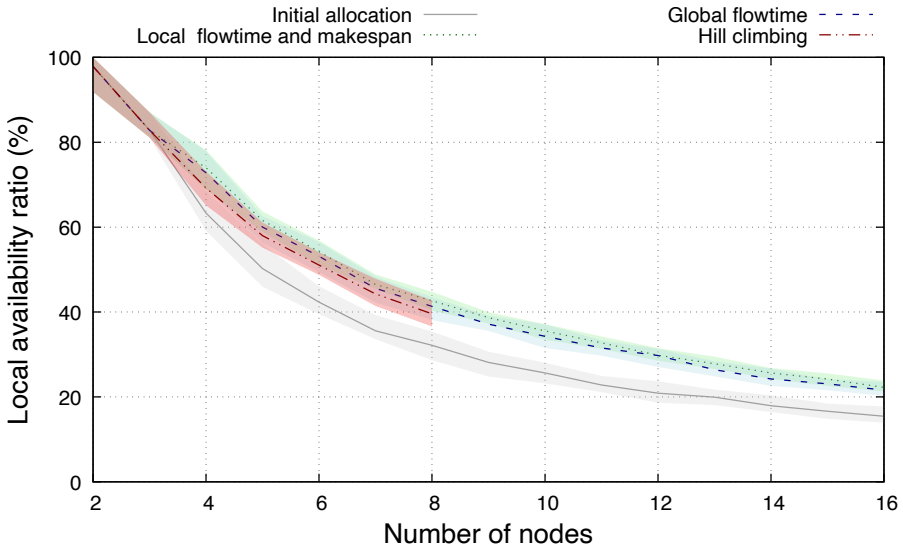


Fig. 6: Local availibility ratio of the initial allocation, the allocations reached by our strategy, by the strategy proposed in [2], and by the hill climbing algorithm

Figure 6 compares the local availibility ratio of the initial allocation, the allocations reached by our strategy, by the strategy proposed in [2] and by the hill climbing algorithm. We observe that the availibility ratios obtained with our stategies or with the hill climbing algorithm are close. Even if, unlike the latter, our stategy does not consider all

the possible unary delegations, it turns out to be efficient by selecting the remote tasks whose delegation decreases the cost.

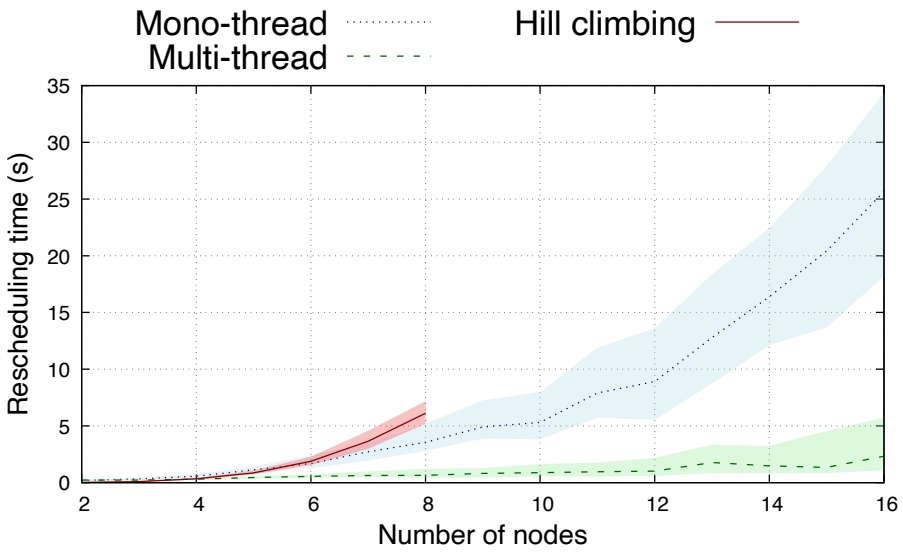


Fig. 7: Rescheduling time of our strategy with one or more threads compared with the rescheduling time of the hill climbing algorithm

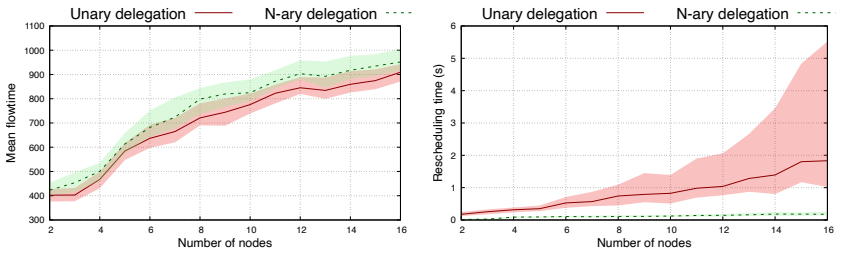
Figure 7 compares the rescheduling time of our strategy with one or more threads and the rescheduling time of the hill climbing algorithm. Since our multi-thread strategy run on several cores, we observe that the speedup increases with the number of agents. By example, with a similar flowtime (if the observable non determinism of the executions is neglected), the multi-thread version is 10 times faster than the mono-thread version for 16 agents.

6.3 *N*-ary delegation

Here, we want to verify that the *n*-ary delegations allow to reduce the flowtime.

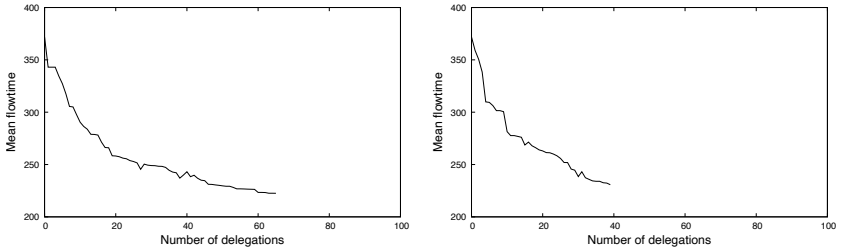
Figures 8a and 8b respectively compare the flowtime and rescheduling time for our unary strategy with our *n*-ary strategy. We observe that, if the flowtime of our *n*-ary delegation is sligthly not as good as than the one of the unary delegation strategy, the gain in terms of rescheduling time gain is much more beneficial.

Figure 9 shows the evolution of the flowtime for both offer strategies for a particular reallocation problem. We observe that the *n*-ary strategy reduces the number of delegations (40 versus 66) required to reach a stable allocation with similar flowtime. Therefore, the *n*-ary strategy reduces the rescheduling time ($0,35$ seconds versus $1,8$ seconds).

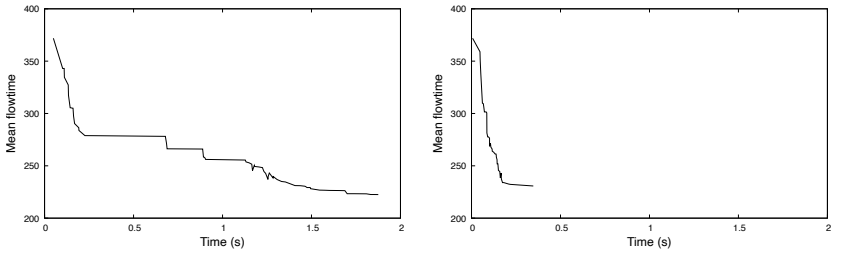


(a) Flowtime (b) Rescheduling time

Fig. 8: Flowtime and rescheduling time for our unary strategy and our n-ary strategy



(a) Flowtime of the unary delegation strategy by number of delegations (b) Flowtime of the n-ary delegation strategy by number of delegations



(c) Flowtime of the unary delegation strategy by time (d) Flowtime of the n-ary delegation strategy by time

Fig. 9: Flowtime of the unary (left) and n-ary (right) delegation strategies by number of delegations (top) and by time (bottom)

6.4 Distributed Constraint Optimization Problem (DCOP)

We want to compare our strategy with a DCOP resolution method to show that: (a) our rescheduling time is much lower, (b) our flowtime is better.

Finding the optimal allocation for a MASTA+ problem with n tasks, m nodes and ℓ jobs (cf. Section 3) can be formalized with:

1. n decision variables x_i such that,

$$x_i = (o-1) \times n + k \text{ if } \tau_i \text{ is the } k^{th} \text{ task starting from the end on } \nu_o \tag{24}$$

2. n^2 constraints ensuring that each task is assigned to a single position

$$\forall i \in [1,n] \; \forall j \in [1,n] \setminus \{i\} \; x_i \neq x_j; \tag{25}$$

3. the objective function to minimize is $C(\overrightarrow{A})$.

We consider here the MGM2 algorithm [17] – Maximum Gain Message – as the most suitable DCOP resolution method, since it is a distributed local search algorithm which is approximate and asynchronous. We used the pyDCOP libray [19]. We consider 100 MASTA+ problems with $m = 2$ nodes, $\ell = 4$ jobs and $n = \ell \times m = 8$ tasks.

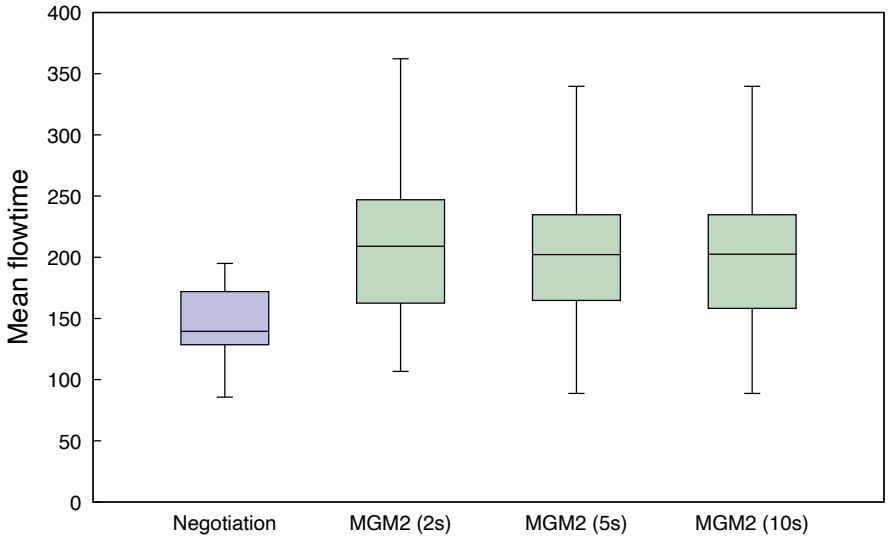


Fig. 10: Box plot of flowtimes reached by our stategy in 55 millisecondes (mean value) and by the MGM2 algorithm with a timeout of 2, 5 and 10 seconds, respectively

Figure 10 compares the flowtimes reached by our stategy in 55 millisecondes (mean value) and by the MGM2 algorithm with a timeout of 2, 5 and 10 seconds, respectively. It is worth noticing that MGM2 never returns a solution when the timeout is 2 seconds.

In this case, we consider that the random initial allocation is returned. Beyond the fact that the rescheduling time can be explained because MGM2 is implemented in Python whereas our strategy runs on a Java Virtual Machine [10], our experiments show that even if the timeout is set to 5 seconds, MGM2 provides an allocation whose flowtime is greater than the one reached by our strategy. Increasing the timeout does not allow to improve the flowtime of the allocation returned by MGM2. We can notice that MGM2 never returns an allocation with $m = 3$ nodes, $\ell = 5$ jobs and $n = 3 \times \ell \times m = 45$ tasks even with a timeout of 15 minutes. This algorithm does not scale for this kind of problems.

7 Conclusion

In this paper, we have proposed a multi-agent strategy for the reassignment of tasks-nodes based on the location of the required resources in order to minimize the mean flowtime of concurrent jobs. We generalized the notion of delegation to consider any bilateral reallocation (delegation or swap of several tasks) and we defined an acceptability criterion of the bilateral reallocations in order to reduce the rescheduling time and the mean flowtime reached by our strategy.

Since our negotiation process continuously adapts the allocation by reducing the completion time of the jobs for the bottleneck agents in order to improve the load-balancing, the flowtime reached by our strategy is similar to the one reached by a classical heuristic approach while significantly reducing the rescheduling time. On the one hand, the consumption strategy performs the cheapest jobs before the most expensive ones. On the other hand, the offer strategy selects a job which can reduce the completion times of the proposer by choosing a receiver which is not a bottleneck for the impacted jobs and by choosing a task whose delegation reduces its cost since it is locally executed. This task selection strategy is repeated by the proposer to build an offer bundle as long as it improves the flowtime. Our experiments show that such n-ary delegations improve the rescheduling time. We have compared our approach with a DCOP resolution method, i.e. the MGM2 algorithm for which the rescheduling time and flowtime are significantly higher.

Beyond the scope of this work, the influence of the replication factor could be investigated in a sensitivity analysis. Since no negotiation is triggered when the agents believe that the allocation is stable, the effort required for negotiation (reasoning and communication) is negligible compared to the benefit of the load-balancing. Due to the local decisions of agents about the next task to delegate/execute, our multi-agent strategy can tackle a large number of tasks, so it is scalable.

Since our negotiation framework allows it, we are considering to add a counter-offer strategy that selects a counterpart for suggesting swaps in order to improve the flowtime. More generally, future works should extend the task reallocation process toward an iterated, dynamic and on-going process, which takes place concurrently with the task execution, allowing the distributed system to be adaptive to disruptive phenomena (task consumption, job release, slowing down nodes).

References

1. Baert, Q., Caron, A.C., Morge, M., Routier, J.C., Stathis, K.: An adaptive multi-agent system for task reallocation in a MapReduce job. Journal of Parallel and Distributed Computing **153**, 75–88 (Jul 2021). https://doi.org/10.1016/j.jpdc.2021.03.008
2. Beauprez, E., Caron, A.C., Morge, M., Routier, J.C.: A Multi-Agent Negotiation Strategy for Reducing the Flowtime. In: Proc. of 13th International Conference on Agents and Artificial Intelligence (ICAART). vol. 1, pp. 58–68 (Feb 2021)
3. Beauprez, E., Morge, M.: Scala implementation of the Extended Multi-agents Situated Task Allocation. https://gitlab.univ-lille.fr/maxime.morge/smastaplus (2020)
4. Beynier, A., Charpillet, F., Szer, D., Mouaddib, A.I.: DEC-MDP / DEC-POMDP. In: Olivier Buffet, O.S. (ed.) Markov Decision Processes in Artificial Intelligence, pp. 277–313. Wiley-ISTE (2010)
5. Bruno, J., Coffman, Jr., E.G., Sethi, R.: Scheduling independent tasks to reduce mean finishing time. Commun. ACM **17**(7), 382–387 (Jul 1974)
6. Chen, B., Potts, C.N., Woeginger, G.J.: Handbook of combinatorial optimization, chap. A review of machine scheduling: Complexity, algorithms and approximability, pp. 1493–1641. Springer (1998)
7. Choi, H.L., Brunet, L., How, J.P.: Consensus-based decentralized auctions for robust task allocation. IEEE transactions on robotics **25**(4), 912–926 (2009)
8. Conway, R., Maxwell, W., Miller, L.: Theory of Scheduling. Addison- Wesley, Reading, MA (1967)
9. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. Journal of Artificial Intelligence Research **61**, 623–698 (2018)
10. Fulgham, B., Gouy, I.: The Computer Language Benchmarks Game. https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html (2021)
11. Horn, W.: Minimizing average flow time with parallel machines. Operations Research **21**(3), 846–847 (1973)
12. Jiang, Y.: A survey of task allocation and load balancing in distributed systems. IEEE Transactions on Parallel and Distributed Systems **27**(2), 585–599 (2016)
13. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval research logistics quarterly **2**(1-2), 83–97 (1955)
14. Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Prasad, M.N., Raja, A., Vincent, R., Xuan, P. Zhang, X.Q.: Evolution of the GPGP/TAEMS domain-independent coordination framework. Autonomous agents and multi-agent systems **9**(1-2), 87–143 (2004)
15. Li, S., Negenborn, R.R., Lodewijks, G.: A Distributed Constraint Optimization Approach for Vessel Rotation Planning. In: Computational Logistics. pp. 61–80. Springer (2014)
16. Lightbend: Akka is the implementation of the actor model on the JVM. http://akka.io (2020)
17. Maheswaran, R.T., Pearce, J.P., Tambe, M.: Distributed algorithms for DCOP: A graphical-game-based approach. In: Bader, D.A., Khokhar, A.A. (eds.) Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems, September 15-17, 2004, The Canterbury Hotel, San Francisco, California, USA. pp. 432–439. ISCA (2004)
18. Rubinstein, A.: Perfect equilibrium in a bargaining model. Econometrica **50**(1), 97–102 (1 1982)
19. Rust, P., Picard, G.: pyDCOP is a python library for Distributed Constraints Optimization. https://github.com/Orange-OpenSource/pyDcop (2021)
20. Schaerf, A., Shoham, Y., Tennenholtz, M.: Adaptive load balancing: A study in multi-agent learning. Journal of Artificial Intelligence Research **2**, 475–500 (1995)

21. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artificial Intelligence **101**(1-2), 165–200 (1998)
22. Turner, J., Meng, Q., Schaefer, G., Soltoggio, A.: Distributed strategy adaptation with a prediction function in multi-agent task allocation. In: Proc. of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS). pp. 739–747 (2018)
23. Wellman, M.P.: A market-oriented programming environment and its application to distributed multicommodity flow problems. Journal of Artificial Intelligence Research **1**, 1–23 (1993)
24. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proc. of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI); San Jose, CA, USA. pp. 15–28 (2012)