A Multi-Agent Negotiation Strategy for Reducing the Flowtime

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

A Multi-Agent Negotiation Strategy for Reducing the Flowtime

Ellie Beauprez[1], [a], Anne-Cécile Caron[1] [b], Maxime Morge[1] [c], and Jean-Christophe Routier[1] [d]

[1] *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*

Keywords:
Multi-Agent Systems, Distributed Problem Solving, Negotiation and Interaction Protocols

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. In this context, we propose a novel strategy based on cooperative agents used to optimize the rescheduling of tasks for multiple jobs submitted by users in order to be executed as soon as possible. It allows an agent to determine locally the next task to process and the next task to delegate according to its knowledge, its own belief base and its peer modelling. The novelty of our strategy lies in the ability of agents to identify opportunities and bottleneck agents, and afterwards to reallocate some of the tasks. Our contribution is that, thanks to concurrent bilateral negotiations, tasks are continuously reallocated according to the local perception and the peer modelling of agents. In order to evaluate the responsiveness of our approach, we implement a prototype testbed and our experimentation reveals that our strategy reaches a flowtime which is close to the one reached by the classical heuristic approach and significantly reduces the rescheduling time.

1 INTRODUCTION

Data science involves the processing of large volumes of data which requires distributed file system and parallel programming. This emerging distributed computing topic brings new challenges related to task allocation and load-balancing. This paper is concerned with a class of practical applications where (a) some jobs, i.e. sets of tasks, are concurrently submitted by users in order to be executed as soon as possible, and (b) some of the resources, e.g. data, required to successfully execute a task are distributed at different network nodes.

In this class of applications, we consider here MapReduce (Dean and Ghemawat, 2004) which is the most prominent distributed data processing model for tackling vast amount of data on commodity clusters as with Hadoop (The Apache Software Foundation, 2020) or Spark (Zaharia et al., 2012). MapReduce jobs are divided into a set of tasks that are distributed on nodes ; the division labour should capitalize upon the way resources are distributed in the system. Indeed, as several resources are necessary to perform a task, any initial allocation inevitably requires fetching some of these resources from other nodes, thus incurring an extra time cost for task execution.

In order to tackle the problem of load-balancing and task reallocation in applications such as those that motivate this work, multi-agent technologies have received a lot of attention (Jiang, 2016). Most of the existing works adopting the market-based approach (Walsh and Wellman, 1998; Shehory and Kraus, 1998; An et al., 2010; Turner et al., 2018) model the load-balancing problem as a non-cooperative game. By contrast, in line with (Baert et al., 2019), we assume that agents are cooperative, viz.: they share the same objective and there is no shared knowledge, including any knowledge about the whole task allocation. Beyond (Baert et al., 2019), we consider here several jobs which are divided

[a] https://orcid.org/0000-0003-3865-5005
[b] https://orcid.org/0000-0001-6672-5686
[c] https://orcid.org/0000-0003-2139-7150
[d] https://orcid.org/0000-0001-8032-6323

into a set of tasks that can be performed by any multi-task agent. The agents aim at minimizing the mean flowtime of several concurrent jobs, i.e. the average completion time of these jobs. The main difficulty lies in the fact that task reallocation does not only change the workload of each agent but also their own scheduling. For this purpose, we propose a negotiation strategy which is composed of a consumption strategy and a delegation strategy. While the consumption strategy allows agents to decide the next tasks to execute, the delegation strategy decides which deal is suggested or accepted. The latter is based on peer modelling and determines the agent behaviour at each point of choice for the negotiation protocol: (a) the offer strategy selects a potential delegation, i.e. a task and a receiver; (b) the acceptability rule determines whether the agent accepts or declines a delegation. Inspired by our practical application, the experiments on our testbed evaluate the responsiveness of our approach.

Specifically, our contributions are as follows:

- We formalize the multi-agent situated task allocation problem with concurrent jobs where tasks have different costs for different agents due to the resource locality.
- We design a multi-agent strategy which allows the agents to continuously identify opportunities and bottleneck agents within a current unbalanced allocation. Afterwards, concurrent and bilateral negotiations amongst agents are triggered to locally delegate some of the tasks. Tasks reallocations occur continuoulsy according to the local perception and the peer modelling of agents.
- We conduct extensive experiments which show that our method reaches a flowtime which is close to the one reached by the classical heuristic approach and significantly reduces the rescheduling time.

The paper is structured as follows. After an overview of the related work in Sec. 2, we formalize the multi-agent situated task allocation problem with concurrent jobs in Sec. 3. Sec. 4 describes the operations of consumption/delegation and the negotiation process. Sec. 5 specifies the consumption strategy, i.e. how agents choose the scheduling order of their own bundle. Sec. 6 details the delegation strategy, i.e. how agents choose which task to negotiate with whom. Our empirical evaluation is described in Sec. 7. Finally, Sec. 8 summarizes our contribution and outlines our future work.

2 RELATED WORK

Classical scheduling problems are constrained optimization problems that can be approximated by different heuristics such as hill climbing or simulated annealing (See (Pinedo, 2008) for a survey). The limitations of these approaches for task reallocation in distributed systems are due to the following aspects:

- Decentralization: global control causes a performance bottleneck as it must collect status information of the entire system in real time. By contrast, agents can take local decisions over an existing allocation in order to improve the load-balancing.
- Adaptation: classical scheduling problems are static. The inaccurate estimation of tasks execution time and the disruptive phenomena (task consumption, job release, etc.), may require major modifications in the existing allocation to stay optimal. Agents can act in dynamic environments that evolve over time.

These are the reasons why multi-agent scheduling has received significant attention for load-balancing problems in distributed systems (See (Jiang, 2016) for a recent survey). Banerjee and Hecker propose in (Banerjee and Hecker, 2017) a general distributed resource allocation protocol for load-balancing coarse-grained jobs on a massively distributed system. They emphasize that the local interactions between agents lead to some complex high-level emergent properties. By contrast, Selvitopi et al. investigate in (Selvitopi et al., 2019) the simultaneous scheduling of fine-grained tasks in order to improve the data locality and balance the workloads. Their approach is based on graph and hypergraph models which make use of application-specific knowledge. Our study aims at bridging this granularity gap by the reassignment of independent tasks within multiple jobs in an application-agnostic way.

Schaerf et al. investigate in (Schaerf et al., 1995) the adaptive behaviour of agents for efficient load-balancing using multi-agent reinforcement learning. Turner et al. combine in (Turner et al., 2018) supervised classification learning with an internal decision-making process for task assignments. Conversely, we do not assume any prior model of the data/environment since

it is not relevant considering the class of practical applications we are concerned with.

In (Jiang and Li, 2011), the authors distinguish two kinds of vicinity: the locality of the resources which are needed to perform the tasks, and the neighbourhood of the agents. On this latter point, we assume a fully connected social network with uniform communication costs. In (Zaharia et al., 2010), the authors are concerned with data locality. In the context of the scheduling problem of MapReduce jobs composed of multiple tasks, they try to place computations near their input data in order to maximize the system throughput. In this paper, we abstract away from the practical applications, but we address a similar challenging problem by considering task consumption, job release and slowing down nodes as unexpected events in a dynamic environment.

On one hand, most of the existing works adopting the market-based approach model the load-balancing problem as a non-cooperative game. For instance, An et al. propose in (An et al., 2010) a distributed negotiation mechanism where selfish agents negotiate over resources both a contract price and a decommitment penalty. On the other hand, most of the solving algorithms for the distributed constraint optimization problems (See (Fioretto et al., 2018) for a recent survey) try to optimize a single-objective function which is assumed to be utilitarian (a sum of costs to be minimized). More recently, (Baert et al., 2019) targets an egalitarian objective which is the minimization of the maximum completion time of the tasks to perform (i.e. the makespan). In this paper, we consider the problem of coordinating agent decisions to find a globally optimal solution for multi-objective functions. The agents try to minimize the mean flowtime of several concurrent jobs, i.e. the mean of the maximum completion times of the tasks in these jobs.

3 SITUATED TASKS

We formalize here the multi-agent situated task allocation problem with concurrent jobs.

A job is a set of independent, non divisible and non preemptive tasks without precedence order. The execution of each task requires resources which are distributed at different nodes. We consider that the 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 having a size (e.g. $|\rho_i|$). The locations of the resources, which are possibly replicated, are determined by the function:*

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

For simplicity, we assume that there is exactly one agent per node (the set of agents is $\mathcal{N}$) and all the resources are accessible by any agent.

The execution of a job (without deadline) consists of the performance of a set of independent tasks which require resources to produce an outcome.

Definition 2 (Job/Task). *Let $\mathcal{D}$ be a distributed system and Outcome 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 $J_i = \{\tau_1, \ldots, \tau_{k_i}\}$ associated to the release date $t^0_{J_i}$ where each task τ is a function which links a set of resources to an outcome: $\tau : 2^{\mathcal{R}} \mapsto$ Outcome.*

$\mathcal{T} = \bigcup_{1 \leq i \leq \ell} J_i$ 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. For the sake of brevity, we denote job(τ) the job containing the task τ. We assume that the number of jobs is negligible with respect to the number of tasks, $|\mathcal{J}| << |\mathcal{T}|$.

The task cost estimates its runtime on a node.

Definition 3 (Task cost). *Let $\mathcal{D}$ be a distributed system and $\mathcal{T}$ be a set of tasks. the cost function is s.t.:*

$$c : \mathcal{T} \times \mathcal{N} \mapsto \mathbb{R}^*_+ \tag{2}$$

The cost of the task τ_i for the node ν_j is a strict positive real number denoted $c(\tau_i, \nu_j)$.

The difficulty lies in the specification of this function in order to have a good estimation of the runtime. 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 ($c(\tau, \nu_i) \leq c(\tau, \nu_j)$) if the required resources are "more local" to ν_i than to ν_j. We postpone the full specification of the task cost function to the experimental setup (Sec. 7).

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

Definition 4 (MASTA+). *A multi-agent situated task allocation problem is a quadruple MASTA+ = $\langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ where:*

- $\mathcal{D} = \langle \mathcal{N}, \mathcal{E}, \mathcal{R} \rangle$ *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 set of ℓ jobs, i.e. a partition of tasks s.t.*

$$(\mathcal{T} = \bigcup_{1 \leq i \leq \ell} J_i) \wedge (J_i \cap J_i = \emptyset \text{ with } 1 \leq i \neq j \leq \ell) \tag{3}$$

- $c : \mathcal{T} \times \mathcal{N} \mapsto \mathbb{R}_+^*$ *is the task cost function.*

A task allocation is a distribution of sorted bundles at different nodes.

Definition 5 (Allocation). *Let MASTA+ be a task allocation problem. An allocation is a vector of m sorted task bundles. $\vec{A} = ((B_1, \prec_1), \ldots, (B_m, \prec_m))$ where 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 ($\prec_i \subseteq \mathcal{T} \times \mathcal{T}$) s.t. $\tau_j \prec_i \tau_k$ means that if $\tau_j, \tau_k \in B_i$ then τ_j is performed before τ_k by ν_i. The allocation $\vec{A}$ is s.t.:*

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

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

All the tasks are assigned (Eq. 4) and each task is assigned to a single node (Eq. 5). For brevity, we denote:

- $\vec{\mathrm{B}}_i = (\mathrm{B}_i, \prec_i)$, 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. the jobs having at least one task in B_i;
- $\nu(\tau, \vec{\mathrm{A}})$, the node responsible for τ in $\vec{\mathrm{A}}$.

The sum of the task costs in a bundle estimates its runtime. Formally, The *workload* of ν_i for $\vec{\mathrm{A}}$ is:

$$\mathrm{w}_i(\vec{\mathrm{A}}) = \sum_{\tau \in \mathrm{B}_i} \mathrm{c}(\tau, \nu_i) \tag{6}$$

In order to evaluate a task allocation, we consider the flowtime, which measures the time between the release date of the jobs and their completion dates, and the makespan which is the completion time of all the jobs.

Definition 6 (Flowtime/Makespan). *Let MASTA+ be a task allocation problem and $\vec{A}$ be an allocation. We define:*

- *the delay of τ for ν_i,*

$$t(\tau, \nu_i) = \sum_{\tau' \in B_i | \tau' \prec_i \tau} c(\tau', \nu_i) \tag{7}$$

- *the completion time of $\tau \in \mathcal{T}$ for $\vec{A}$,*

$$C_\tau(\vec{A}) = t(\tau, \nu(\tau, \vec{A})) + c(\tau, \nu(\tau, \vec{A})) \tag{8}$$

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

$$C_J(\vec{A}) = \max_{\tau \in J} \{C_\tau(\vec{A})\} \tag{9}$$

- *the completion date of $J \in \mathcal{J}$ for $\vec{A}$,*

$$t_J^E(\vec{A}) = t_J^0 + C_J(\vec{A}) \tag{10}$$

- *the mean flowtime of $\mathcal{J}$ for $\vec{A}$,*

$$C(\vec{A}) = \frac{1}{\ell} \sum_{J \in \mathcal{J}} C_J(\vec{A}) \tag{11}$$

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

$$C_{max}(\vec{A}) = \max_{J \in \mathcal{J}} t_J^E(\vec{A}) \tag{12}$$

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

$$L(\vec{A}) = \sum_{\tau \in \mathcal{T}} \frac{\sum_{\rho \in \mathcal{R}_\tau,\ \nu(\tau, \vec{A}) \in l(\rho)} |\rho|}{\sum_{\rho \in \mathcal{R}_\tau} |\rho|} \tag{13}$$

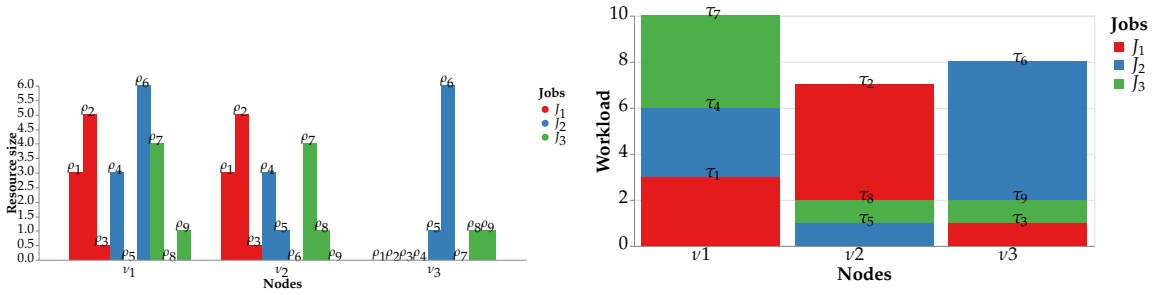
The delay of a task for a node estimates the runtimes (costs) of the previous tasks in the bundle (Eq. 7) since we assume the nodes are never inactive. The completion time of a task is the sum of the delay before the task is started plus its runtime (Eq. 8). Contrary to the cost of the task, its delay and its completion time depend on the scheduling order over the bundle. According to Eq. 9, a job is finished for all the agents when they have performed all the tasks in this job. The completion date of a job is the finishing date (Eq. 10). The mean flowtime is the mean completion time of the jobs (Eq. 11). Since the makespan measures the maximum completion time of the jobs (Eq. 12), it is the maximum workload of the nodes:

$$\mathrm{C}_{max}(\vec{\mathrm{A}}) = \max_{\nu_i \in \mathcal{N}} \{\mathrm{w}_i(\vec{\mathrm{A}})\} \tag{14}$$

Contrary to the makespan, the flowtime depends on the scheduling order. The local availability ratio measures the proportion of locally processed resources (Eq. 13).

Example 1 (MASTA+). *We consider $\mathcal{D} = \langle \mathcal{N}, \mathcal{E}, \mathcal{R} \rangle$ where:*

- *3 nodes $\mathcal{N} = \{\nu_1, \nu_2, \nu_3\}$ are fully connected $\mathcal{E} = \{(\nu_1, \nu_2), (\nu_1, \nu_3), (\nu_2, \nu_3)\}$;*
- *9 resources $\mathcal{R} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7, \rho_8, \rho_9\}$ are replicated over 2 nodes (Fig. 1a). For instance, ρ_1, with $|\rho_1| = 6$, is over ν_1 and ν_2 but not over ν_3.*



(a) Resource distribution (b) *Allocation*

Figure 1: The resource distribution and the allocation sorted by the LCJF strategy

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

Table 1: The cost of the tasks for the nodes

We also consider MASTA+ $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *with:*

- 9 *tasks* $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8, \tau_9\}$. *Each one requires the corresponding resource, e.g.* ρ_1 *is required for* τ_1 ;
- 3 *jobs* $\mathcal{J} = \{J_1, J_2, J_3\}$, *s.t.* $J_1 = \{\tau_1, \tau_2, \tau_3\}$, $J_2 = \{\tau_4, \tau_5, \tau_6\}$ *and* $J_3 = \{\tau_7, \tau_8, \tau_9\}$.
- *the cost function which is represented in Tab. 1. We assume that the cost of a task is proportional to the resource size and two times more important if the resource is remote.*

The allocation represented in Fig. 1b is s.t. $\vec{B}_1 = (\tau_1, \tau_4, \tau_7)$, $\vec{B}_2 = (\tau_5, \tau_8, \tau_2)$ *and* $\vec{B}_3 = (\tau_3, \tau_9, \tau_6)$. *The makespan and the mean flowtime are* $C_{max}(\vec{A}) = 10$ *and* $C(\vec{A}) = 8.33$, *respectively.*

In summary, we consider heterogeneous nodes because the cost of tasks depend on the processing nodes due to the locality of the underlying resources. Our objective is to minimize the mean flowtime of the concurrent jobs containing several tasks:

$$R_m|\Sigma_{J \in \mathcal{J}} \max_{\tau \in J} C_\tau(\vec{A})$$

4 CONSUMPTION AND DELEGATION PROCESS

We describe here the operations of consumption/delegation and we sketch the negotiation protocol.

The addition/removal of the task τ in the bundle $\vec{B}_i$ modifies not only the set of tasks but also the scheduling order over the bundle. Formally,

- if $\tau \notin B_i$ then $\overrightarrow{B_i \oplus \tau}$ denotes the bundle which contains the set of tasks $B_i \cup \{\tau\}$ sorted with $\prec_i$;
- if $\tau \in B_i$ then $\overrightarrow{B_i \ominus \tau}$ denotes the bundle which contains $B_i \setminus \{\tau\}$ sorted with $\prec_i$.

These operations imply a rescheduling of the bundle. As we will see in Sec. 5, the consumption strategy specify the scheduling order.

A task consumption is a disruptive event which modifies not only the allocation of tasks but also the underlying problem.

Definition 7 (Consumption). *Let MASTA+* $= \langle \mathcal{D}, \mathcal{T}, \mathcal{J}, c \rangle$ *be a task allocation problem and* $\vec{A}$ *be an allocation. The consumption of* τ *by* ν_i *with a non-empty bundle* ($B_i \neq \emptyset$) *leads to the allocation* $\vec{A}' = \gamma(\nu_i, \vec{A})$ *for the problem* $MASTA' = \langle \mathcal{D}, \mathcal{T}', \mathcal{J}', c \rangle$ *where:*

$$\mathcal{T}' = \mathcal{T} \setminus \{\min_{\prec_i} B_i\} \tag{15}$$

$$\mathcal{J}' = \begin{cases} \mathcal{J} \setminus \{job(\min_{\prec_i} B_i)\} \\ \quad \text{if } job(\min_{\prec_i} B_i) = \{\min_{\prec_i} B_i\} \\ \mathcal{J} \quad \text{else} \end{cases} \tag{16}$$

In the latter case:

$$J'_j = \begin{cases} J_j \setminus \{\min_{\prec_i} B_i\} & \text{if } job(\min_{\prec_i} B_i) = J_j \\ J_j & \text{else} \end{cases} \tag{17}$$

and

$$\vec{B}'_j = \begin{cases} \overrightarrow{B_i \ominus \min_{\prec_i} B_i} & \text{if } j = i \\ \vec{B}_j & \text{else} \end{cases} \tag{18}$$

The task is removed from the set of tasks (Eq. 15) and from the corresponding job (Eq. 17). The latter is removed if it only contains τ (Eq. 16). The task is removed from the allocation (Eq. 18). Obviously, a task consumption can increase neither the makespan nor the flowtime but it decreases the local makespan ($w_i(\gamma(\nu_i,\vec{A})) < w_i(\vec{A})$) and the local flowtime ($\Sigma_{J\in jobs(B_i)} C_J(\gamma(\nu_i,\vec{B}_i)) < \Sigma_{J\in jobs(B_i)} C_J(\vec{B}_i)$). The sequence of consumptions, which consists of an iteration of MASTA+ problems, removes one by one all the tasks from the initial allocation to the empty one.

A task delegation is a disruptive event which changes the current allocation, i.e. a reallocation.

Definition 8 (Delegation)**.** *Let MASTA+ $= \langle \mathcal{D},\mathcal{T},\mathcal{J},c\rangle$ be a task allocation problem and $\vec{A} = (\vec{B}_1,\ldots,\vec{B}_m)$ be an allocation. If the task τ is assigned to the donor ν_i ($\tau \in B_i$), then the delegation to the recipient ν_j leads to the allocation $\delta(\tau,\nu_i,\nu_j,\vec{A})$ with the m bundles $\delta(\tau,\nu_i,\nu_j,\vec{B}_k)$ such as:*

$$\delta(\tau,\nu_i,\nu_j,\vec{B}_k) = \begin{cases} \overrightarrow{B_i \ominus \tau} & \text{if } k = i \\ \overrightarrow{B_j \oplus \tau} & \text{if } k = j \\ \vec{B}_k & \text{else} \end{cases} \quad (19)$$

A task delegation must reduce the local makespan and the local flowtime.

Definition 9 (Socially rational delegation)**.** *Let MASTA+ $= \langle \mathcal{D},\mathcal{T},\mathcal{J},c\rangle$ be a task allocation problem, $\vec{A}$ be an allocation and $\delta(\tau,\nu_i,\nu_j,\vec{A})$ the allocation after the delegation of τ by ν_i to ν_j. This delegation is socially rational:*

- *with respect to the makespan if and only if the local makespan decreases*

$$w_j(\vec{A}) + c(\tau,\nu_j) < w_i(\vec{A}) \quad (20)$$

- *with respect to the flowtime if and only if the local flowtime decreases*

$$\Sigma_{J\in jobs(B_i\cup B_j)} \max(C_J(\overrightarrow{B_i \ominus \tau}), C_J(\overrightarrow{B_j \oplus \tau})) < \Sigma_{J\in jobs(B_i\cup B_j)} \max(C_J(\vec{B}_i), C_J(\vec{B}_j)) \quad (21)$$

An allocation is stable if none of the agents have socially rational delegations. In a stable allocation with respect to the makespan, agents cannot locally improve the makespan.

Property 1 (Termination)**.** *Let MASTA+ $= \langle \mathcal{D},\mathcal{T},\mathcal{J},c\rangle$ be a task allocation problem and $\vec{A}$ be a non-stable allocation with respect to the makespan. There is always a finite path of socially rational delegations with respect to this criterion, which leads to a stable allocation with respect to makespan.*

The property derives from Theo. 7 in (Endriss et al., 2006). By contrast, a sequence of socially rational delegations with respect to the flowtime does not necessarily lead to a stable allocation for this criterion.

Agents operate in multiple bilateral single-round negotiations for task delegations. Each negotiation, which is based on the alternating offers protocol (Rubinstein, 1982), includes three decision steps: (a) the offer strategy of the proposer (Sec. 6) which selects a potential delegation, i.e. a task in its bundle and a recipient, (b) the acceptability rule (Sec. 6) allows the responder to determine whether it rejects or accepts such a delegation, and (c) in the latter case, the delegation is confirmed or withdrawn by the proposer depending on the interleaved consumptions.

5 CONSUMPTION STRATEGY

We describe here the consumption strategy adopted by an agent to select the next task to perform, i.e. a scheduling order over its bundle based on its local knowledge.

We consider a task allocation problem MASTA+ $= \langle \mathcal{D},\mathcal{T},\mathcal{J},c\rangle$ and an allocation $\vec{A}$. Each agent $\nu_i \in \mathcal{N}$ which knows the problem and has a local perception of the allocation, i.e. its bundle $\vec{B}_i$, can deduce the following metrics:

- its workload (Eq. 6);
- the delay of the tasks in its bundle (Eq. 7);
- the completion time of these tasks (Eq. 8);
- the completion time of the jobs for the node,

$$C_J(\vec{B}_i) = \begin{cases} \max_{\tau\in J\cap B_i}\{C_\tau(\vec{B}_i)\} & \text{if } J \in jobs(B_i) \\ 0 & \text{else} \end{cases} \quad (22)$$

- the job costs for the node,

$$c(J,\nu_i) = \Sigma_{\tau\in J\cap B_i} c(\tau,\nu_i) \quad (23)$$

It is worth noticing that the mean completion time of the jobs for an agent depends on the scheduling order over its bundle.

A consumption strategy is defined by a scheduling order, i.e. a strict total order over the bundle. Since we aim at minimizing the completion time of jobs rather than tasks we focus

on job-oriented consumption strategies. Such a strategy is a lexicographic order which consists of sorting first jobs, and then the tasks inside the same job.

Definition 10 (Job-oriented strategy)**.** *Let $\lhd_i$ and $\blacktriangleleft_i$ two strict total order relations over $\mathcal{T}$ and $\mathcal{J}$, respectively. A job sorting strategy based on $(\lhd_i, \blacktriangleleft_i)$ sorts the bundle B_i according to the strict total order relation $\prec_i$ defined by:*

$$\begin{aligned}&\forall \tau_j, \tau_k \in B_i \ \tau_j \prec_i \tau_k \Leftrightarrow \\ &job(\tau_j) \blacktriangleleft_i job(\tau_k) \vee (job(\tau_j) = job(\tau_k) \wedge \tau_j \lhd_i \tau_k)\end{aligned} \tag{24}$$

$J_1 \blacktriangleleft_i J_2$ means that the tasks in J_1 are prior to the tasks in J_2. The tasks in the same job are consecutive in the bundle.

The job-oriented consumption strategy we consider in this paper aims at minimizing the completion times of jobs by sorting them according to their costs.

Definition 11 (LCJF)**.** *Let $\lhd_i$ a strict total order on $\mathcal{T}$. A job sorting strategy based on $(\lhd_i, \blacktriangleleft_i)$ is said "Locally Cheapest Job First" (LCJF) if and only if the relation $\blacktriangleleft_i$ over the jobs satisfies :*

$$\begin{aligned}&\forall J_j, J_k \in \mathcal{J},\ J_j \blacktriangleleft_i J_k \Leftrightarrow \\ &c(J_j, \nu_i) < c(J_k, \nu_i) \vee (c(J_j, \nu_i) = c(J_k, \nu_i) \wedge J_j < J_k)\end{aligned} \tag{25}$$

where $<$ denotes the order over the jobs induced by the natural order over their identifiers.

This strategy executes the tasks in the cheapest jobs before the tasks in the expensive ones.

The relation over the jobs $\blacktriangleleft_i$ is based on the knowledge of the agent about the MASTA+ problem and its own bundle. This scheduling order is strict and total since $<$ discriminates the ties. Since we assume a strict total order over the tasks $\lhd_i$ (e.g. arbitrary the natural order over the task identifiers), $\prec_i$ is also strict and total.

A LCJF strategy allows ν_i to locally minimize the completion time of jobs. None permutation of jobs in the bundle can strictly decrease the local flowtime.

Lemma 1 (LCJF)**.** *Let $\prec_i$ be a LCJF strategy based on $(\lhd_i, \blacktriangleleft_i)$.*

$$\begin{aligned}&\forall \sigma \in S(jobs(B_i)), \\ &\Sigma_{J \in jobs(B_i)} C_J(\vec{B}_i) \leq \Sigma_{J \in jobs(B_i)} C_J(\sigma(\vec{B}_i))\end{aligned} \tag{26}$$

where $S(jobs(B_i))$ denotes the set of all the permutations of the jobs assigned to ν_i.

The lemma derives from Theo. 3.1.1 in Chap. 3 of (Pinedo, 2008).

Example 2 (LCJF)**.** *Let us consider Ex. 1. If the bundles are $B_1 = \{\tau_1, \tau_4, \tau_7\}$, $B_2 = \{\tau_2, \tau_5, \tau_8\}$ and, $B_3 = \{\tau_3, \tau_6, \tau_9\}$, the allocation sorted with the LCJF strategy (Fig. 1b) is s.t.:*

- $\vec{B}_1 = (\tau_1, \tau_4, \tau_7)$ *since* $(c(J_1, \nu_1) = c(\tau_1, \nu_1)) < (c(J_2, \nu_1) = c(\tau_4, \nu_1)) < (c(J_3, \nu_1) = c(\tau_7, \nu_1))$*;*
- $\vec{B}_2 = (\tau_5, \tau_8, \tau_2)$ *since* $(c(J_2, \nu_2) = c(\tau_5, \nu_2)) = (c(J_3, \nu_2) = c(\tau_8, \nu_2)) < (c(J_1, \nu_2) = c(\tau_2, \nu_2))$ *and* $J_2 < J_3$*;*
- $\vec{B}_3 = (\tau_3, \tau_9, \tau_6)$ *since* $(c(J_1, \nu_3) = c(\tau_3, \nu_3)) = (c(J_3, \nu_3) = c(\tau_9, \nu_3)) < (c(J_2, \nu_3) = c(\tau_6, \nu_3))$ *and* $J_1 < J_3$*.*

6 DELEGATION STRATEGY

We describe here the different parts of the delegation strategy (peer modelling, acceptability rule, offer strategy) and we sketch the agent behaviour.

The **peer modelling** is built upon the information exchanged by the agents through messages. In particular, the agent ν_i informs its peers about the job costs for it ($c(J, \nu_i)$, $\forall J \in \mathcal{J}$) before the negotiation process and after each delegation in which it is involved. Since the number of jobs is much smaller than the number of tasks, the size of theses messages ($\mathcal{O}(|\mathcal{J}|)$) is negligible wrt the task bundle descriptions ($\mathcal{O}(|\mathcal{T}|)$). The peer modelling of the target ν_j by the subject ν_i is based on:

- the belief base of the subject, eventually partial or obsolete, which contains the belief about the job costs for ν_j ($c^i(J, \nu_j)$ $\forall J \in \mathcal{J}$) and the belief about the workload of ν_j ($w_j^i(\vec{A}) = \Sigma_{J \in \mathcal{J}} c^i(J, \nu_j)$);
- the consumption strategy of the target assumed by the subject which boils down to the relation over the jobs adopted by the target, denoted $\blacktriangleleft_j^i$.

For readability, we denote $c^i(J, \nu_i) = c(J, \nu_i)$ and $w_i^i(\vec{A}) = w_i(\vec{A})$. From its beliefs about the target and the assumed consumption strategy, the subject can deduce the completion time of the job J for the target, eventually after the addition/removal of τ : $C_J{}^i(\vec{B}_j)$, $C_J{}^i(\overrightarrow{B_j \oplus \tau})$ and $C_J{}^i(\overrightarrow{B_j \ominus \tau})$. Then, the subject can deduce the

completion time of a job for the allocation:

$$\mathrm{C_J}^i(\vec{\mathrm{A}}) = \max_{\nu_j \in \mathcal{N}} \mathrm{C_J}^i(\vec{\mathrm{B}}_j) \text{ where } \mathrm{C_J}^i(\vec{\mathrm{B}}_i) = \mathrm{C_J}(\vec{\mathrm{B}}_i) \tag{27}$$

The subject considers the target as a bottleneck for the job J ($\nu_j = \mathrm{nodeMax}^i(\vec{\mathrm{A}}, \mathrm{J})$) if the completion time of this job for the target is the completion time for the allocation, i.e. $\mathrm{C_J}^i(\vec{\mathrm{A}}) = \mathrm{C_J}^i(\vec{\mathrm{B}}_j)$.

The **acceptability rule** is a local decision by the recipient of a delegation, which is based on its knowledge and its peer modelling, to determine whether a delegation is accepted or declined depending on its perception of the social rationality of the delegation.

Definition 12 (Acceptability criterion)**.** *Let $\delta(\tau, \nu_i, \nu_j, \vec{A})$ be the delegation of τ from ν_i to ν_j in $\vec{A}$. This delegation is acceptable by the recipient :*

- *with respect to the makespan if and only if the recipient believes that the delegation decreases the local makespan,*

$$w_j(\vec{A}) + c(\tau, \nu_j) < w_i^j(\vec{A}) \tag{28}$$

- *with respect to the flowtime if and only if the recipient believes that the delegation decreases the local flowtime,*

$$\Sigma_{J \in \mathcal{J}} \max(C_J^j(\overrightarrow{B_i \ominus \tau}), C_J(\overrightarrow{B_j \oplus \tau})) < \Sigma_{J \in \mathcal{J}} \max(C_J^j(\vec{B}_i), C_J(\vec{B}_j)) \tag{29}$$

While the first criterion is built upon the belief about the workload of the donor (Eq. 28), the second one is built upon the knowledge about the completion times of the jobs for the recipient before/after the delegation and its belief about the completion times of the jobs for the donor before/after the delegation (Eq. 29).

In order to reduce the flowtime, the acceptability rule consists in checking not only the acceptability criterion wrt the flowtime but also the acceptability criterion wrt the makespan which guarantees the convergence of the negotiation process (Prop. 1).

The **offer strategy** of a possible donor ν_i, which is based on its knowledge and its peer modelling, selects a potential delegation: a recipient, a task and so a job. The strategy is divided in 4 steps:

1. **Job selection.** According to the principle of subsidiarity, the goal of the donor is to reduce the completion time of the jobs it is responsible for. In order to reduce not only the completion time of one job for the donor but also the completion time of the next jobs in $\vec{\mathrm{B}}_i$, our heuristic selects the prior job for which the donor is a bottleneck,

$$\forall \mathcal{J}' \subseteq \mathcal{J},\ \mathrm{J}_* = \sigma_i(\mathcal{J}') = \min_{\blacktriangleleft_i} \{\mathrm{J} \in \mathrm{jobs}(\mathrm{B}_i) \cap \mathcal{J}' \mid \nu_i = \mathrm{nodeMax}^i(\vec{\mathrm{A}}, \mathrm{J})\} \tag{30}$$

2. **Recipient selection.** The jobs assigned to the recipient which are impacted by the delegation are those after J_* according to $\blacktriangleleft_j^i$. Not to increase the completion times of these jobs, our heuristic selects the recipient for whom the sum of the differences between the completion time for the allocation and the completion time for this agent is the greatest one,

$$\forall \mathcal{N}' \subseteq \mathcal{N},\ \nu_* = \sigma_i(\mathcal{N}', \mathrm{J}_*) = \min_{<} \{\operatorname*{argmax}_{\nu_j \in \mathcal{N}'} \sum_{\mathrm{J}_* \blacktriangleleft_j^i \mathrm{J}} (\mathrm{C_J}^i(\vec{\mathrm{A}}) - \mathrm{C_J}^i(\vec{\mathrm{B}}_j))\} \tag{31}$$

where $<$ denotes the order over the nodes induced by the natural order over their identifiers.

3. **Task selection.** In order to reduce the completion times, the donor selects a distant task whose delegation reduces its cost since it is locally executed. For this purpose, our heuristic selects the task in the job J_* or in the prior jobs in $\vec{\mathrm{B}}_i$ with the highest payoff. In case of tie, the prior task is chosen,

$$\forall \mathcal{T}' \subseteq \mathcal{T},\ \tau_* = \sigma_i(\mathcal{T}', \nu_*, \mathrm{J}_*) = \min_{\lhd_i} \{\operatorname*{argmax}_{\tau \in \mathcal{T}' \cap \mathrm{B}_i \cap \{\mathrm{J} \mid \mathrm{J} = \mathrm{J}_* \vee (\mathrm{J} \blacktriangleleft_i \mathrm{J}_*)\}} \mathrm{c}(\tau, \nu_i) - \mathrm{c}(\tau, \nu_*)\} \tag{32}$$

4. **Validation**. By symmetry with the acceptability criterion (Def. 12), the trigger criterion is a local decision by the donor to determine whether a delegation is perceived as socially rational. In order to warranty the convergence of the negotiation process, the trigger rule is a conjunction of the trigger criterion wrt the makespan and the trigger criterion wrt the flowtime. If the trigger rule is not satisfied, then another task ($\mathcal{T}' = \mathcal{T}' \setminus \{\tau_*\}$ in step 3) or another recipient ($\mathcal{N}' = \mathcal{N}' \setminus \{\nu_*\}$ in step 2), eventually another job ($\mathcal{J}' = \mathcal{J}' \setminus \{\mathrm{J}_*\}$ in step 1) is chosen. In case of failure, no delegation is proposed and the agent enters in pause until its belief base is updated and a new opportunity (i.e. a potential delegation) is found.

In our approach, the task reallocation is the outcome of negotiations between agents adopting the same **behaviour**: they alternatively play the roles of proposer, responder and contractor. The agents execute this behaviour according to their knowledge and beliefs. The agent behavior is specified in (Beauprez and Morge, 2020) by a deterministic finite state automaton[1]. In order to avoid deadlock, the proposals are associated with deadlines.

It is worth noticing the reception of the message from the peers updates the belief base of the agent and none proposal is sent when the agents believes that the allocation is stable.

Example 3 (Delegation strategy)**.** *Let us consider the MASTA+ instance and the allocation $\vec{A}$ of Ex. 1. We assume that the agents have up-to-date beliefs and that they know that all of them adopt the LCJF strategy. Contrary to ν_1 and ν_2, the agent ν_3 can make a proposal since it selects:*

1. *the job for which it is a limiting factor (Eq. 30), $J_* = J_2$;*
2. *the less bottleneck agent for the impacted jobs (Eq. 31) which are the jobs J_3 for ν_1 and the jobs J_1 and J_3 for ν_2:*

$$\sum_{J_* \blacktriangleleft_1^3 J} (C_J^3(\vec{A}) - C_J^3(\vec{B}_1)) = 0 \tag{33}$$

$$\sum_{J_* \blacktriangleleft_2^3 J} (C_J^3(\vec{A}) - C_J^3(\vec{B}_2)) = 8 \tag{34}$$

   Therefore, $\nu_* = \nu_2$;
3. *the prior task with the highest payoff (Eq. 32) in J_2 or in the prior jobs, i.e. J_1 and J_3:*

$$\begin{aligned}(c(\tau_6,\nu_3) - c(\tau_6,\nu_2)) &= 6 - 12 = -6\\ (c(\tau_9,\nu_3) - c(\tau_9,\nu_2)) &= 1 - 2 = -1\\ (c(\tau_3,\nu_3) - c(\tau_3,\nu_2)) &= 1 - 0.5 = 0.5\end{aligned} \tag{35}$$

   Therefore, $\tau_* = \tau_3$;
4. *the delegation is triggerable since the trigger criterion wrt the flowtime is satisfied,*

$$\Sigma_{J\in\mathcal{J}} \max(C_J(\overrightarrow{B_3 \ominus \tau_3}), C_J^3(\overrightarrow{B_2 \oplus \tau_3})) = 16.5 < \Sigma_{J\in\mathcal{J}} \max(C_J(B_3), C_J^3(B_2)) = 17.0 \tag{36}$$

   and the trigger criterion wrt the makespan is satisfied,

$$w_2^3(\vec{A}) + c(\tau,\nu_2) = 7.5 < w_3(\vec{A}) = 8.0 \tag{37}$$

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

The donor ν_3 delegates the task τ_3 to the agent ν_2 to reach $\vec{A}' = \delta(\tau_3,\nu_3,\nu_2,\vec{A})$ s.t. $\vec{B}_1' = \{\tau_1,\tau_4,\tau_7\}$, $\vec{B}_2' = \{\tau_5,\tau_8,\tau_3,\tau_2\}$ and $\vec{B}_3' = \{\tau_9,\tau_6\}$. The allocation is stable.

7 EMPIRICAL EVALUATION

We consider as a practical application the distributed deployment of the MapReduce design pattern in order to process large datasets on a cluster, as with Hadoop (The Apache Software Foundation, 2020) or Spark (Zaharia et al., 2012). 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.:

$$\begin{aligned} c_i(\tau,\nu_j) &= \sum_{\rho_j \in \mathcal{R}_\tau} c_i(\rho_j,\nu_j)\\ \text{with } c_i(\rho_j,\nu_i) &= \begin{cases} |\rho_j| \text{ if } \nu_i \in l(\rho_j) \\ \kappa \times |\rho_j| \text{ else} \end{cases}\end{aligned} \tag{38}$$

where we empirically calibrate $\kappa = 2$ as a realistic value.

To our best knowledge, no alive project of agent platforms provides a logic-based agent programming language, allowing a direct implementation of concepts such as beliefs and goals, that supports the development of cognitive agents for large-scale, distributed applications and services. That is the reason why our testbed (Beauprez and Morge, 2020) is implemented with the general purpose programming language Scala and Akka (Lightbend, 2020) 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.

We consider three metrics: (1) the mean flowtime (Eq. 11), (2) the local availability ratio (Eq. 13), and (3) the scheduling time. We aim at (i) comparing the allocation reached by our negociation process with the classical approach, and (ii) evaluating the acceleration thanks to the decentralization.

The outcome reached by the general-purpose solver IBM® ILOG® CPLEX® tackling the underlying non-linear discrete mathematical optimization problem is poor. This is the reason why

our baseline is a hill climbing algorithm. The latter and our method both start with the same randomly generated initial allocation sorted according to the LCJF strategy. At each step, the hill climbing algorithm iteratively selects among all the possible delegations, the one which minimizes the *flowtime* after the application of the LCJF strategy.

The MASTA+ problem instances we consider are such that $m \in [2;12]$ nodes/agents, $\ell \in [2;5]$ jobs and $n = 3 \times \ell \times m$ tasks. We consider 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.

The hypothesis we want to test are: (1) the flowtime reached from our strategy is close to the one reached by the classical approach and (2) the decentralization significantly reduces the scheduling time.

Figures 2a and 2b present the medians of the flowtime and the scheduling time exhibited by the methods depending on the number of nodes and the number of jobs. Figures 2c, 2d and 3 focus on the medians and the standard deviations of all the metrics depending on the number of nodes with $\ell = 4$ jobs. It is worth noticing that the hill climbing algorithm has been used with small MASTA+ instances due to its prohibitive scheduling time. At each step, the hill climbing algorithm considers all the possible delegations, thus it reaches an allocation with a better flowtime than our strategy. Since the overhead of our strategy is 25%, our delegation strategy seems to be efficient even if the acceptability criterion wrt the makespan, which is required to guarantee the convergence, may lead to discard some delegations which may reduce the flowtime (Sec. 4). This is due to the fact that the delegation strategy selects the distant tasks whose delegations reduce their cost in order to improve local availability ratio which is slightly better than the one reached by the hill climbing algorithm. Moreover, since the latter evaluates at each step all the possible delegations, its scheduling time is much higher than the one of our negociation strategy. For instance, it is six times higher for 9 agents and 4 jobs.

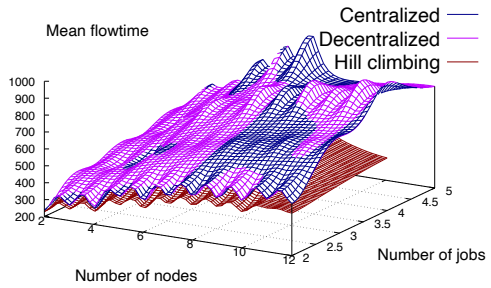
It is worth noticing that the gap between the scheduling times of the two methods increases exponentially with the number of agents, while the gap between the flowtimes is roughly constant. We can expect a higher scheduling time if we adopt a local search method such as the simulated annealing without any warranty about the optimality of the 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).

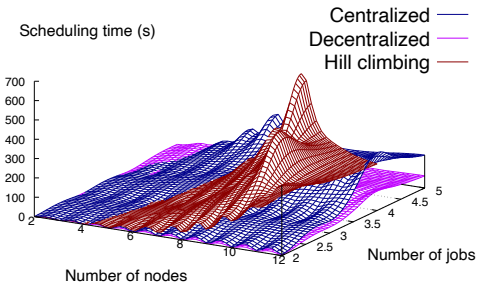
Finally, we observe that, when it is decentralized on several cores, the acceleration of our algorithm increases with the number of agents and jobs. For instance, with a similar *flowtime* neglecting the observable nondeterminism of their executions, the decentralized version runs 3 times faster than the centralized one for 12 agents and 4 jobs.

8 CONCLUSION

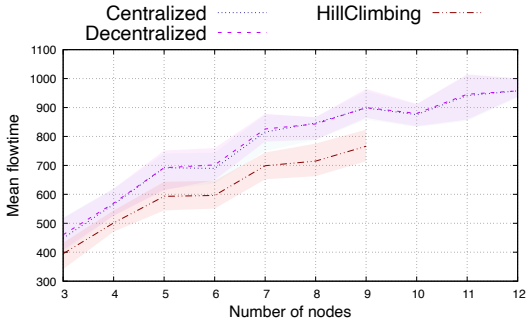
In this paper, we have proposed a multi-agent system for task reallocation among distributed nodes based on the location of the required resources in order to minimize the mean flowtime of concurrent jobs. Our prototype has been empirically evaluated. Our experiments show that the flowtime reached by our strategy is close to the one reached by the classical heuristic approach and it significantly reduces the rescheduling time. This is due to the fact that our negotiation process continuously adapts the allocation in order to improve the load-balancing by reducing the completion times of the jobs for the bottleneck agents. On one hand, the consumption strategy performs the tasks of the cheapest jobs before the most expensive ones. On the other hand, the delegation strategy selects a job which can reduce the completion times of the donor 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. A sensitivity analysis to study the influence of the replication factor has been beyond the scope of this work, but it is certainly worth of further investigation. Obviously, our approach is scalable since it tackles a large number of tasks due to the local decisions of agents about the next task to delegate/execute. Moreover, the overhead of the negotiation is negligible with respect to the benefit of the load-balancing since no negotiation is triggered when the agents believe that the allocation



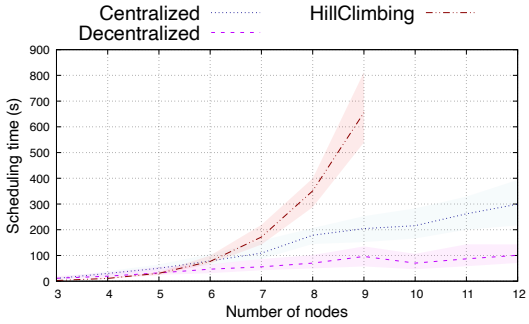
(a) *Flowtime*



(b) *Scheduling time*



(c) *Flowtime*



(d) *Scheduling time*

Figure 2: The mean flowtime and the scheduling time for our (centralized and decentralized) negociation strategy and the hill climbing algorithm.

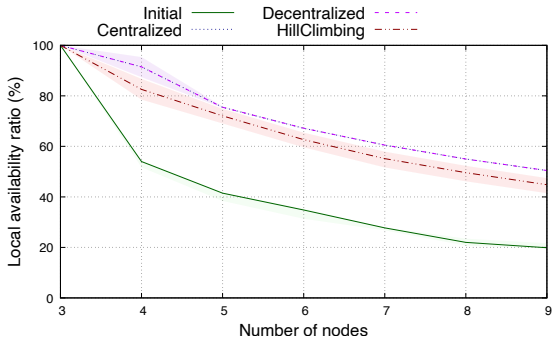


Figure 3: The local availability ratio for our (centralized and decentralized) negociation strategy, the hill climbing algorithm and the initial allocation.

is stable.

A comparative study of our strategy with different distributed resolution methods algorithms is beyond the scope of this paper, but it is certainly worth of further investigation. Some of our experiments suggest that we need to extend our negotiation framework to consider (a) task swaps to improve the mean flowtime of stable allocations, and (b) a less restrictive acceptability rule which currently discards some task delegations which may reduce the flowtime. Generally, future work must extend the task reallocation 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).

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their stimulating comments which help us to improve the paper.

REFERENCES

An, B., Lesser, V., Irwin, D., and Zink, M. (2010). Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proc. of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 981–988.

Baert, Q., Caron, A.-C., Morge, M., Routier, J.-C., and Stathis, K. (2019). A Location-Aware Strategy for Agents Negotiating Load-balancing. In *Proc.*

of 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, Oregon, United States.

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

Beauprez, E. and Morge, M. (2020). Scala implementation of the Extended Multi-agents Situated Task Allocation. https://gitlab.univ-lille.fr/maxime.morge/smastaplus.

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the 9th Symposium on Operating Systems Design and Implementation*, pages 137–150.

Endriss, U., Maudet, N., Sadri, F., and Toni, F. (2006). Negotiating Socially Optimal Allocations of Resources. *Journal of Artificial Intelligence Research*, 25:315 – 348.

Fioretto, F., Pontelli, E., and Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698.

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

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

Lightbend (2020). Akka is the implementation of the actor model on the JVM. http://akka.io.

Pinedo, M. L. (2008). *Scheduling. Theory, Algorithms, and Systems. Third Edition*. Springer.

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

Schaerf, A., Shoham, Y., and Tennenholtz, M. (1995). Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500.

Selvitopi, O., Demirci, G. V., Turk, A., and Aykanat, C. (2019). Locality-aware and load-balanced static task scheduling for MapReduce. *Future Generation Computer Systems*, 90:49–61.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200.

The Apache Software Foundation (2020). Apache Hadoop. https://hadoop.apache.org.

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

Walsh, W. E. and Wellman, M. P. (1998). A market protocol for decentralized task allocation. In *Proc. of the 3rd International Conference on Multiagent Systems (ICMAS)*, pages 325–332.

Zaharia, M., Borthakur, D., Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. (2010). Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of the EuroSys 2010 Conference*, pages 265–278.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI); San Jose, CA, USA*, pages 15–28.