A Location-Aware Strategy for Agents Negotiating Load-balancing

Quentin Baert, Anne-Cécile Caron, Maxime Morge and Jean-Christophe Routier
Univ. Lille, CNRS, Centrale Lille
UMR 9189 - CRIStAL
F-59000 Lille, France
Email: {Quentin.Baert,Anne-Cecile.Caron,Maxime.Morge,Jean-Christophe.Routier}@univ-lille.fr
Kostas Stathis
Department of Computer Science, Royal Holloway, University of London, Egham TW20 0EX, UK
Email: Kostas.Stathis@rhul.ac.uk

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

I. Introduction

The problem of load balancing and task allocation in distributed systems manifests itself in many applications such as cloud computing, peer-to-peer and ad-hoc networks, pervasive computing, online social networks and big data processing. In this work we are concerned with a class of such practical applications where (a) some of the resources, e.g. data, required to successfully execute a task are distributed at different network nodes, and (b) some of the nodes may encounter potential execution hazards, e.g. slowing down nodes or communication lags. 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 [1]. In this class of applications the task allocation can be challenged during the task execution and should capitalize upon the way resources are distributed in the system.

In order to tackle the problem of load balancing and task allocation in applications such as those that motivate this work, multi-agent technologies have received a lot of attention, especially those based on market-based scheduling, for example see [2], [3], [4]. Apart from decentralization, i.e. avoiding performance bottlenecks due to global control, we show here that a multi-agent approach for situated task allocation supports two additional crucial requirements (a) concurrency – where task reallocation and task executions are concurrent, and (b) adaptation – where task reallocation is triggered when a disruptive event is performed. We assume that agents are cooperative, viz., they share the same objective: minimizing the global runtime. We also assume there is no shared knowledge, including any knowledge about the whole task allocation. However, agents have a model of their peers, i.e. they are able to compute the cost of tasks for their peers. We further assume that a task can be performed by any single agent without preemption and precedence order. A task is indivisible, without deadline and not shareable i.e. a task belongs to only one agent at a time.

The work formalizes the multi-agent situated task allocation problem. By adopting a market-based approach, we totally decentralize the load-balancing, whereby cooperative agents try to minimize the completion time of the last task to perform, i.e. the makespan. For this purpose, we propose a novel location-aware strategy for agents to perform the load-balancing. Our work significantly extends [5] with a formal framework for a negotiation strategy and its full experimental results that are in line with [6], where the locality of resources is taken into account. When agents identify opportunities within a current unbalanced allocation, they trigger concurrent and one-to-many negotiations to locally reallocate some tasks. The tasks are reallocated according to the proximity between the required resources and the nodes executing the tasks. This dynamic and on-going task reallocation process takes place concurrently with the task execution and so the distributed system is adaptive to disruptive phenomena (task consumption, slowing down nodes).

We consider as a practical application the distributed deployment of the MapReduce design pattern in order to process large datasets on a cluster [7], as with Hadoop [8]. Even if some skews in real-world datasets lead to poor load-balancing [9], a centralized scheduler cannot be used as a baseline to compare with our approach due to the large number of tasks. However, multi-agent negotiation allows task reallocation during the reduce phase in order to decrease the job runtime. Our approach is validated for such an application since the preliminary empirical results show that the location-aware strategy allows to improve the overall runtime of the data processing.

The paper is structured as follows. After an overview of the related work in Sec. II, we formalize the multi-agent situated task allocation problem and we define in Sec. III the socially rational task delegation considered by the agents in order to locally improve the task allocation. Sec. IV sketches the iterated negotiation process which is concurrent with task consumptions. Sec. V describes the location-aware strategy, i.e. how agents choose which task to perform/negotiate. Our practical application and empirical validation are described in Sec. VI. Finally, Sec. VII summarizes our contribution and outlines our future work.

II. Related work

Classical scheduling problems have been the subject of extensive research producing offline schedulers for some simple models [10]. The problem of minimizing the makespan (the completion time of the last task to perform) with n tasks on m unrelated machines (i.e. with different capabilities), denoted $R||C_{max}$, is NP-hard [11]. Pseudo-polynomial algorithms developed for this problem include: the earliest completion time heuristic [12] (ECT), the local search heuristics [13], the branch and bound algorithm [14] and two-phase heuristics based on linear programming [15]. Even if the ECT heuristic is an approximation algorithm which gives acceptable results with very small computational requirements, centralized algorithms cannot be applied to our scenario with a large number of tasks (i.e. $82,283$ keys in Sec. VI). The location-aware strategy is a decentralized local search heuristic.

Multi-agent scheduling [1] has received significant attention for load-balancing problems in distributed systems, but it is different from the classical scheduling problems due to the following requirements:

- Scalability: global control causes a performance bottleneck as it must collect status information of the entire system in real time. Instead, task allocation can be negotiated by agents representing the nodes [2], [16].
- Responsiveness: classical scheduling problems are static. The inaccurate estimation of tasks execution time, made worse by disruptive phenomena (task consumption, slowing down nodes, etc.), may require major modifications in the existing allocation to stay optimal [4]. Rather than continuously recomputing an optimal allocation, some local modifications during the tasks processing can improve the load-balancing [17], [18], [19].

Most of the existing works adopting the market-based approach (e.g. [2], [20]) model the load-balancing problem as a non-cooperative game in order to optimize user-centric metrics rather than system-centric ones such as the global runtime we consider in this paper. Contrary to our work, [3] consider task assignments to groups of agents as necessary since tasks cannot be performed by a single reliable agent. In this paper, we assume that a task, which can be performed by any single agent without preemption and precedence order, is indivisible, not shareable (i.e. a task belongs to only one agent at a time) and without deadline. [21] has studied multi-agent resource allocation but their work is restricted to the assignment of a single resource per agent. We assume here that a bundle of tasks can be assigned to each agent. In the continuity of [22], [23], [24] proposes potentially distributable solving methods using agent-based negotiation for resource allocation. Even if the network topology is out of the scope of our work (agents are fully connected in our practical application), we go a step further by effectively decentralizing the negotiation, and by challenging this allocation during task processing.

For the processing of real-world datasets by the distributed deployment of the MapReduce design pattern, it is well known that data skews are widespread and they can lead to poor load-balancing [9]. In particular the partitioning skew occurs when one reducer processes a larger number of keys than others. Since the job ends when all the reduce tasks are finished, the job runtime is penalized by the most loaded reducer. This data skew is tackled by [25], [26] using parametrization based on prior knowledge about the data and the distributed computing environment. We address this issue with the dynamic and adaptative task reallocation which is concurrent with the task consumption. Thus, reallocation is adaptative to the data processing. This enables us to tackle the following real-world issues: (a) the lack of prerequisite knowledge over the data, (b) the inaccurate estimation of task execution time, and (c) the execution hazards (slowing down nodes, communication lag). To the best of our knowledge, no other proposal is scalable and responsive, like ours.

III. Situated task allocation

We formalize here the multi-agent situated task allocation (MASTA) problem where tasks have different costs (runtime) for different agents due to the resource locality.

Definition 1 (MASTA): A multi-agent situated task allocation problem of size (k, m, n) with $k \geq 1$, $m \geq 2$ and $n \geq 1$ is a tuple $MASTA = \langle Node, \mathcal{A}, \mathcal{T}, l, d, c \rangle$ s.t.:

- $Node = \{node_1, \ldots, node_k\}$ is a set of k nodes;
- $\mathcal{A} = \{1, \ldots, m\}$ is a set of m agents;
- $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is a set of n tasks to perform;
- $l : \mathcal{A} \mapsto Node$ is a function which returns the location of an agent;
- $d : \mathcal{T} \times Node \mapsto \mathbb{N}^+$ is a function which specifies how many resources for a task are on a particular node;

- $c : \mathcal{T} \times Node \mapsto \mathbb{R}_+^*$ specifies the cost of a task τ at a given location such that the tasks with more local resources are cheaper:

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

In the rest of the paper, l_i, $c_i(\tau)$ and $d_i(\tau)$ denote $l(i)$, $c(\tau, l_i)$ and $d(\tau, l_i)$, respectively. Similarly, we denote $d_\tau = \sum_{node \in Node} d(\tau, node)$. We say that τ is local, partially local, or distant for agent i if $d_i(\tau) = d_\tau$, $d_i(\tau) < d_\tau$, or $d_i(\tau) = 0$, respectively.

We consider a particular MASTA problem and we evaluate the task allocation from a collective viewpoint by considering the maximum completion time, i.e. the makespan.

Definition 2 (Task allocation/Workload/Makespan): A task allocation P is a partition of tasks among agents, i.e a set of m task bundles $\{P(1), \ldots, P(m)\}$ such that:

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

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

The workload of the agent $i \in \mathcal{A}$ in the allocation P is defined as:

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

The makespan of P is defined as:

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

We define the socially rational task delegation considered by the agents in order to locally improve the task allocation.

Definition 3 (Socially rational task delegation): Let P be a task allocation. The delegation δ of the task τ from agent i to agent j is defined s.t. the resulting allocation $\delta(P) = \{P'(1), \ldots, P'(m)\}$ is as follows:

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

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

The delegation is socially rational iff:

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

Since a socially rational task delegation δ strictly decreases the local makespan between the two agents, it does not increase the global makespan ($C_{max}(\delta(P)) \leq C_{max}(P)$). We can now denote $\Gamma_i(P)$ the set of socially rational task delegations that an agent i can initiate:

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

A task allocation P is said stable if no agent can perform a socially rational task delegation.

Property 1: A non-stable task allocation can always lead to a stable one using a finite number of socially rational task delegations.

Proof 1: Let P a task allocation and $W_P =< w_{i_1}, \ldots, w_{i_m} >$ the vector of the workloads in decreasing order where w_{i_r} denotes the r^{th} largest workload. If P is not stable, there exists a socially rational task delegation δ which leads to $P' = \delta(P)$. Formally,

$$\exists i, j \in \mathcal{A}, max\big(w_i(P'), w_j(P')\big) < max\big(w_i(P), w_j(P)\big) \quad (10)$$

$$\wedge \forall k \in \mathcal{A} \setminus \{i, j\}, w_k(P) = w_k(P') \quad (11)$$

It implies that $W_{P'} < W_P$ in the lexicographic order. Formally,

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

Since there is a finite number of allocations and the order is strict, there is a finite number of socially rational task delegations.

Let us consider the following walk-through example.

Example 1: Let $MASTA^{ex} =< Node, \mathcal{A}, \mathcal{T}, l, d, c >$ a problem of size $(2, 2, 7)$, where $Node = \{node_1, node_2\}$, $\mathcal{A} = \{1, 2\}$ and $\mathcal{T} = \{\tau_1, \ldots, \tau_7\}$ with $l(1) = node_1$, $l(2) = node_2$. The locations and costs of tasks are represented in Tab. I. Let us consider the task allocations $Pmks$

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

TABLE I: Locations and costs of tasks

and P such that $Pmks = \{\{\tau_1, \tau_3, \tau_5, \tau_6\}, \{\tau_2, \tau_4, \tau_7\}\}$ and $P = \{\{\tau_2, \tau_4, \tau_6\}, \{\tau_1, \tau_3, \tau_5, \tau_7\}\}$. $Pmks$ is optimal since $C_{max}(Pmks) = 35$ ($w_1(Pmks) = 33$ and $w_2(Pmks) = 35$). It is not the case of P since $w_1(P) = 46$, $w_2(P) = 27$, and so $C_{max}(P) = 46$.
Let $P' = \delta_1(P)$ be the task allocation where δ_1 is the delegation of τ_6 from agent 1 to agent 2. Since $w_1(P') = 38$ and $w_2(P') = 40$, δ_1 improves the makespan (i.e. $C_{max}(P') = 40$). However P' is not stable.
Let $P'' = \delta_2(P')$ be the task allocation where δ_2 is the delegation of τ_1 from agent 2 to agent 1. Even if P'' is not optimal ($C_{max}(P'') > C_{max}(Pmks)$), P'' is stable.

IV. Negotiation process

We sketch here the iterated negotiation process which is concurrent with task consumptions. When a task is performed, it is removed from the set of tasks, and so the multi-agent system tries to minimize the makespan with respect to this new MASTA problem.

A task consumption is a disruptive event which modifies the MASTA problem and the task allocation.

Definition 4 (Task consumption): Let P be the current task allocation for the problem $MASTA = \langle Node, \mathcal{A}, \mathcal{T}, l, d, c \rangle$. The consumption γ of the task τ by agent i leads to the task allocation $P' = \gamma(P)$ for the problem $MASTA' = \langle Node, \mathcal{A}, \mathcal{T}', l, d, c\rangle$ such that:

$$\mathcal{T}' = \mathcal{T} \setminus \{\tau\} \tag{12}$$
$$P'(i) = P(i) \setminus \{\tau\} \tag{13}$$
$$\forall j \in \mathcal{A} \setminus \{i\}, P'(j) = P(j) \tag{14}$$

Obviously, task consumption may decrease the makespan. The sequence of task consumptions, which removes all the tasks from the initial allocation until a final empty one $\perp$, consists of an iteration of MASTA problems.

Decentralized task delegation process. Agents operate in concurrent, one-to-many and single-round negotiations for task delegations. Each negotiation, which is based on the Contract Net Protocol [27], includes three decision steps: (a) the choice of the task to negotiate by the strategy of the initiator described in Sec. V, (b) the refusals/bids from the peers based on the social rationality of the task delegation, and (c) the selection of the winning bid by the initiator, e.g. the bidder with the smallest workload. Several negotiations involving different groups of agents may concurrently occur and agents can bid in several simultaneous negotiations as in [28]. The concurrency of task delegations improves the responsiveness of the task reallocation. Even if an agent which is involved in a negotiation as a bidder cannot initiate another negotiation (and conversely), agents can simultaneously bid in concurrent negotiations. In order to tackle the eager bidder problem [29], we adopt a conservative approach where the task delegations are warranted to be socially rational [30]. Therefore, several task delegations can be concurrently negotiated.

Since there is no shared knowledge, an agent has partial and not necessarily true beliefs about the current allocation P. Indeed, agent i knows its own workload $w_i(P)$ and it has a belief base:

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

where $w_j^i(P)$ is the belief of agent i about the workload of agent j in the allocation P.

The set of potential socially rational task delegations $\Gamma_i^{\mathcal{B}}(P)$ that an agent i can initiate in the task allocation P is based on its belief base $\mathcal{B}_i(P)$. Formally,

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

Similarly, the computation of the local makespan by the initiator of a negotiation is based on its belief base, possibly inaccurate. This is the price to pay for the decentralization. However, an agent informs its peers about their workload when they are triggered for the first time, within the negotiation messages, and after each task consumption. Therefore, the bidder belief base is updated and the latter rejects the task delegations which are not socially rational. Therefore, a successful negotiation can only reach a socially rational task delegation, and so tends to improve the makespan.

Concurrent consumptions and delegations. Task delegations and task consumptions are concurrent and complementary since a task removal may allow new socially rational task delegations. Fig. 1 represents their impact on the allocation until all of the tasks are performed. Starting from the initial allocation P_0, agents perform socially rational task delegations to improve the makespan (e.g. the path from P_0 to P_k) until a task consumption (e.g. the edge from P_k to P_0'), which eventually interrupts the path toward a stable allocation (e.g. the path from P_k to P_0' represented in grey). A task consumption may occur when the agents have reached a stable allocation (e.g. P'_{stable}) or not (e.g. P_k).

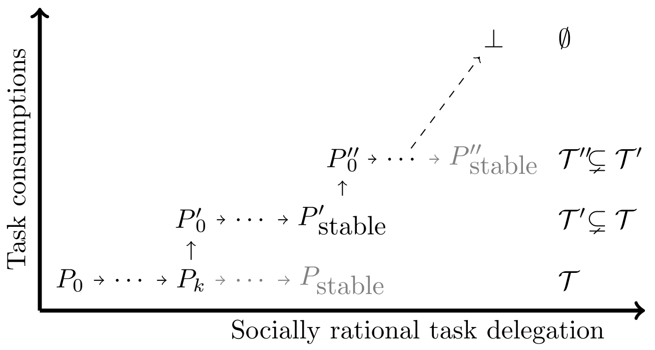
Task consumptions
$\perp$
$\emptyset$
$P_0'' \rightarrow \cdots \rightarrow P''_{\text{stable}}$
$\mathcal{T}'' \subsetneq \mathcal{T}'$
$P_0' \rightarrow \cdots \rightarrow P'_{\text{stable}}$
$\mathcal{T}' \subsetneq \mathcal{T}$
$P_0 \rightarrow \cdots \rightarrow P_k \rightarrow \cdots \rightarrow P_{\text{stable}}$
$\mathcal{T}$
Socially rational task delegation

Fig. 1: Concurrent task consumptions (vertical edges) and task delegations (horizontal edges).

V. Location-aware strategy

During the overall process, task negotiations and task consumptions are concurrent. The strategy of an agent allows to select the next potential socially rational delegation and the next task to perform.

Definition 5 (Strategy): Let P be an allocation, the strategy of agent i is the couple $(perform_i, negotiate_i)$ where:

- $perform_i : 2^{\mathcal{T}} \mapsto \mathcal{T} \cup \{\perp\}$, selects the next task to perform within $P(i)$ or none (denoted $\perp$) if $P(i) = \emptyset$;
- $negotiate_i : 2^{\mathcal{T}} \mapsto \mathcal{T} \cup \{\perp\}$, selects the next task to negotiate within $\Gamma_i^{\mathcal{B}}(P)$ or none (denoted $\perp$) if $\Gamma_i^{\mathcal{B}}(P) = \emptyset$.

Local agnostic strategy. In a first approach, an agent i can perform the largest task in its bundle and negotiate the smallest one in the set of potential socially rational delegations. Formally,

$$\overset{>}{\tau} = \underset{\tau \in P(i)}{\text{argmax}}\{c_i(\tau)\} \tag{17}$$
$$\overset{<}{\tau} = \underset{\tau \in \Gamma_i^{\mathcal{B}}(P)}{\text{argmin}}\{c_i(\tau)\} \tag{18}$$

where $\overset{>}{\tau}$ and $\overset{<}{\tau}$ denote the next task to perform and the next one to negotiate, respectively. This strategy only requires that the agent sorts the tasks by their costs.

However, resource fetching consumes time and represents an extra cost for task execution.

In order to measure the locality of tasks, we define the local availability ratio of an agent i for a task as the ratio between the number of local resources for this task with respect to the agent and the total number of resources for the task:

Definition 6 (Local availability ratio): The local availability ratio of the agent i for the task τ (denoted $o_i(\tau)$) is defined as:

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

The maximum local availability ratio for the task τ is:

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

Location-aware strategy. Intuitively, an agent should perform first the tasks which may cost more for its peers and it should negotiate first the tasks which may cost less for its peers. According to this strategy, an agent performs first the large local tasks and it negotiates first the large distant ones based on its local beliefs and knowledge, i.e. the local availability ratios and the task costs. This strategy is built on a data structure, called local-aware bundle.

The **local-aware bundle** of agent i (see Fig. 2) is divided in three subbundles in accordance with the local availability ratios of the agent for the tasks.

1) *The maximum local bundle* (denoted MB) contains the tasks such that agent i owns at least one resource and there is no other agent which owns more resources for this task. Formally,

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

In MB, the tasks are sorted in decreasing order of cost (cf. left of Fig. 2).

2) *The intermediate local bundle* (denoted IB) contains the tasks which are partially local. Formally,

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

In IB, the tasks are sorted in decreasing order of local availability ratio and the tasks with the same local availability ratio are sorted in decreasing order of cost (cf. center of Fig. 2).

3) *The distant bundle* (denoted DB) contains the tasks which are distant. Formally,

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

In DB, the tasks are sorted in increasing order of cost (cf. right of Fig. 2).

When an agent looks for a task to perform, it starts from the top of the maximum local bundle, i.e. the largest local task. When an agent looks for a task to negotiate, it starts from the bottom of the distant bundle (i.e. the largest distant task) and it selects the first one which is a potential

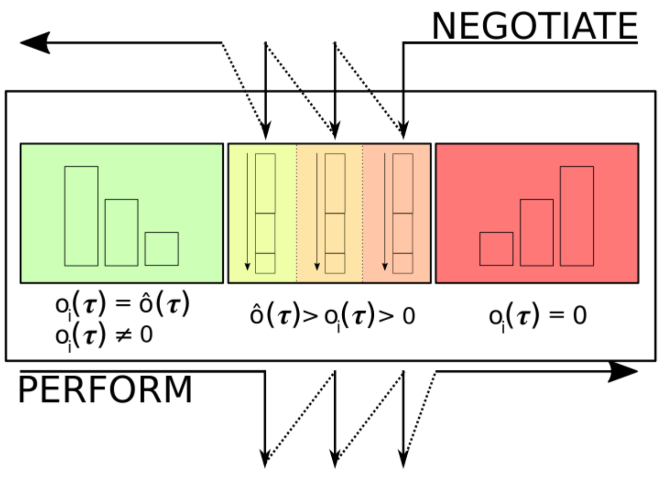

Fig. 2: The local-aware bundle contains the maximum local bundle (at left), the intermediate local bundle (at center) and the distant bundle (at right). The rectangle size represents the cost of the task. The arrows depict the order in which an agent looks for a task to perform/negotiate.

socially rational delegation according to its beliefs, $\mathcal{B}_i(P)$. Formally,

- An agent looks for the task to perform $\overset{\geq}{\tau} \in MB$ s.t.

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

When $MB = \emptyset$, it looks for $\overset{\geq}{\tau} \in IB$ s.t.

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

Then, when $IB = \emptyset$, it looks for $\overset{\geq}{\tau} \in DB$ s.t.

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

Finally, when $MB = IB = DB = \emptyset$, the agent has no task to perform.

- An agent looks for the task to negotiate $\overset{\leq}{\tau} \in DB$ s.t.

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

Then, when $DB = \emptyset$, it looks for $\overset{\leq}{\tau} \in IB$ s.t.

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

Then, when $IB = \emptyset$, it looks for $\overset{\leq}{\tau} \in MB$ s.t.

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

Finally, when such a task is not found the agent stops to negotiate.

It is worth noticing that the leximax order over the intermediate local bundle promotes the principle that an agent should perform large local tasks and negotiate large distant ones. When an agent browses the intermediate local bundle, it starts from the tasks with a high local availability ratio in order to perform them and it starts from the tasks with a low local availability ratio in order to negotiate them. When the agent considers tasks which have the same local availability ratio, it always browses them in the same order: from the costliest to the less costly task (cf. Fig. 2). Thus, when the agent browses the intermediate local bundle to look for a task to perform, it considers the most local and costliest task. When the agent browses the intermediate local bundle to look for a task to negotiate, it considers the less local and costliest task.

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

TABLE II: Local availability ratios

Example 2 (Local-aware bundle): Recall Ex. 1 where we assume, for illustration purposes, that a remote resource is twice more expensive than a local one. Formally,

$$c_i(\tau) = d_i(\tau) + 2\big(d_\tau - d_i(\tau)\big) = 2d_\tau - d_i(\tau) \tag{30}$$

Tab. II represents the local availability ratios of the agents for all the tasks. Let us consider the allocation where agent 1 owns all the tasks. According to the local agnostic strategy, agent 1 cannot discriminate amongst τ_2 and τ_6 for delegation since these tasks have the same cost for the agent. According to the location-aware strategy and its bundle given in Fig. 3, agent 1 considers:

- the execution of the tasks τ_6, τ_1, $\underline{\tau_4, \tau_3,}$ τ_5, τ_2, then τ_7;
- the negotiation of the tasks τ_7, τ_2, τ_5, $\underline{\tau_4, \tau_3,}$ τ_1, then τ_6.

The location-aware strategy leads agent 1 to perform τ_6 and to negotiate τ_7 in order to decrease the makespan since τ_6 is almost local and τ_7 is distant. In particular, agent 1 executes τ_6 before τ_1 since τ_6 is more expensive (cf. Tab. I). We emphasize that the tasks in the intermediate local bundle are sorted first by local availability ratio, then by cost. Whether agent 1 is seeking to negotiate or perform a task, it will consider the tasks which have the same local availability ratio in the same order (e.g. τ_4, then τ_3).

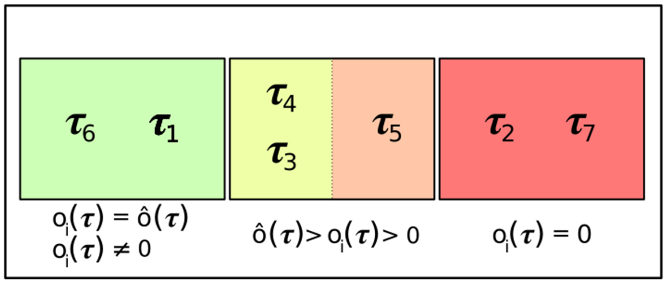

Fig. 3: The local-aware bundle of agent 1

VI. Practical application

We apply our framework to revisit the distributed deployment of the MapReduce design pattern in order to process large datasets on a cluster [7]. Within such a cluster a MapReduce job consists of two successive phases: map and reduce. During the map phase, nodes filter in parallel input data and generate key-value pairs written into data chunks. During the reduce phase, nodes process in parallel the keys and their individual lists of values. In order to transfer map output to the corresponding reduce tasks, the partitioning specifies that all the values for each key are grouped together and make sure that all the values of a single key go to the same reducer. This partitioning is fixed *a priori*; in Hadoop it is evaluated modulo the number of reducers by default.

Even though Hadoop distributes keys evenly across reducers, some reducers are still assigned more data because the assigned key groups contain significantly more values [9]. Thus, the reducers' runtime is highly uneven. By revisiting MapReduce with our location-aware strategy we allow agents to perform the load-balancing and thus improve the job runtime. For this purpose, we specify the cost function in conformance with Eq. 1.

Definition 7 (Locality-based linear cost function): The locality-based linear cost function for agent $i \in \mathcal{A}$ is defined s.t.:

$$c_i(\tau) = \sum_{\rho \in chunks_\tau} c_i(\rho), \text{ with } c_i(\rho) = \begin{cases} |\rho| \text{ if } l_\rho = l_i \\ \kappa \times |\rho| \text{ otherwise} \end{cases} \tag{31}$$

where $chunks_\tau$ denotes the d_τ data chunks $(\rho_1, \ldots, \rho_{d_\tau})$ required by task τ, $|\rho|$ is the size of chunk ρ, and l_ρ its location. We arbitrarily use $\kappa = 10$. The ingenuousness and inaccuracy of the cost function is removed by the dynamic and adaptive task reallocation which allows us to best exploit the computation nodes.

We have developed a multi-agent version of the MapReduce pattern in a distributed system setting using the MAS4Data test-bed [31]. MAS4Data is implemented in Akka [32] for highly concurrent, distributed, and resilient message-driven applications. Even if fault-tolerance of nodes is out of the scope of this paper, we assume that: (a) the message transmission delay is arbitrary but not negligible, (b) the messages sent between a given pair of agents is such that their order is always preserved and (c) messages may be lost. For (c) we have included acknowledgments and deadline mechanisms in the interaction protocol used between agents. In addition, to decrease the complexity related to the design of a reducer agent, we have adopted a modular agent architecture that allows for concurrent operation between negotiation and performance of reduce tasks, as well as separation between communicative and decision making behaviours.

Our experiments have been performed on 16 PCs with 4 cores Intel(R) i7 and 16GB RAM each. They are based on a 8 Gio dataset (82,283 keys) which has been generated such that the initial task allocation (cf. Fig. 5) is poorly load-balanced in order to check that the proximity between the data resources and the processing nodes has an impact on the makespan and so on the job runtime. The initial task allocation is the outcome of the partitioning of the MapReduce process. In other words, when there is no negotiation, MAS4Data has the same behaviour as Hadoop.

Fig. 4 compares the median job runtimes when the agents adopt the local agnostic strategy or the location-aware one with 10 runs for each (re)allocation. We observe that the location-aware strategy significantly improves the runtime, around -7.6%. For comparison, the job runtime without challenging the Hadoop default partitioning is 853 seconds (around $+100\%$). We deduce that the price of the negotiation can be neglected regarding the impact of the load-balancing. Moreover, the negotiation allows to decrease the job runtime and even more with the location-aware strategy.

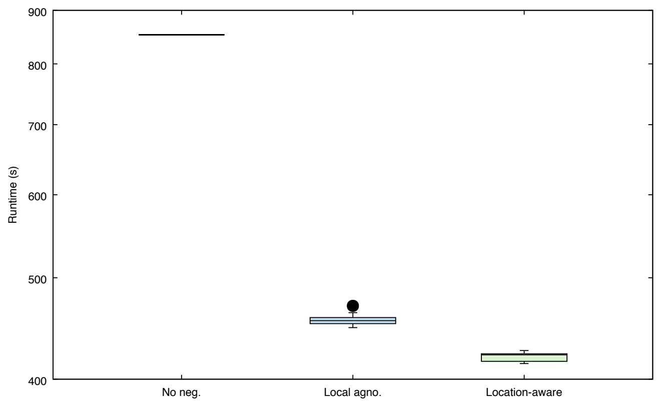

Fig. 4: The median job execution times (and the standard deviations) with the trivial strategy and the locality-based one (10 runs).

Due to the observable nondeterminism of distributed execution, we consider a distinctive runtime. Fig. 5 compares the task allocation when all the tasks have been performed independently of whether they have been negotiated (or not) amongst the 16 agents in accordance with the local agnostic strategy or the location-aware one[1]. Both strategies are compared *ex-post*. We observe that the makespan of the initial allocation is approximately 3.3 10^8, around 2.5 10^8 (-24%) for the negotiation with the local agnostic strategy and 2 10^8 (-30.7%) for the location-aware one. We deduce that the cost of the negotiation, in particular with the location-aware strategy, is less than the time saved by the load-balancing.

Fig. 6 depicts the number of tasks t such that $\alpha \leq o_i(t) < \alpha + 0.1$ where agent i performs t. The initial task allocation is poorly load-balanced since there are more

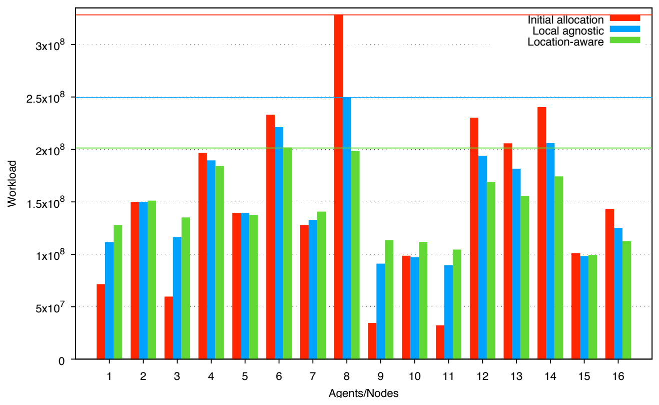

Fig. 5: Initial task allocation (Hadoop) and *ex-post* task allocation with the local agnostic strategy and the location-aware one.

than 4.6 10^4 tasks such as $o_i(\tau) = 0$. We observe that the location-aware strategy promotes the performance of tasks which are mostly local, i.e. $o_i(\tau) \geq 0.5$. This is not the case for the local agnostic strategy. For instance, there are 171 tasks that have been performed by an agent which owns 60% of the chunks (i.e. resources) using the location-aware strategy but none with the local agnostic one. Since the location-aware strategy promotes the execution of tasks by the nodes which are closer to the chunks, the load-balancing is improved and so the job runtime is decreased.

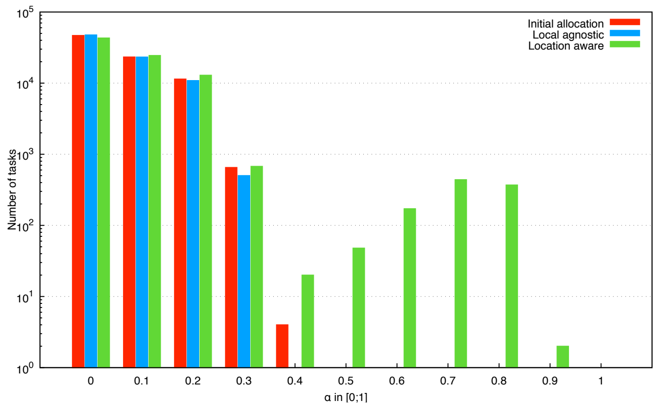

Fig. 6: Number of tasks (with a logarithmic scale) per local availability ratio of the performing agent.

VII. Conclusion

In this paper, we have formalized the multi-agent situated task allocation problem to develop a location-aware strategy for addressing the load balancing problem in distributed systems. The strategy adopts a market-based approach that allows us to decentralize load-balancing in order to minimize the makespan, i.e. the completion time of the last task to perform. During the overall process, task negotiations allow agents that use the strategy to reallocate the tasks concurrently with task consumption. The

[1]Our preliminary empirical analysis shows that: (a) information status (meta-communication) is free with respect to resource fetching in the context of large dataset processing and (b) there is no added value to have more than one reducer agent per node/disk.

strategy also enables agents to select the next potential socially rational delegation and the next task to perform. According to their local beliefs and knowledge, the agents perform first large local tasks and they negotiate first large distant ones. In order to validate our approach, we have developed a prototype where agents negotiate the reduce tasks of the MapReduce design pattern in a distributed setting. Our experimental results show that, for such an application, the proposed location-aware strategy improves the application's load-balancing and so the job execution time.

We are currently evaluating our prototype with numerous and representative experiments with real-word datasets with various jobs/clusters. Some of these settings suggest that we need to extend our strategy to negotiate (a) task swaps to improve the makespan of stable allocations, and (b) task bundles in order to speedup the negotiation process. Even if our adaptive and dynamic approach tackles the problem of drop of performance, our long-term future work should take into account fault tolerance and so data replication.

Acknowledgment

This work is supported by the call ULille "Internationalisation Actions bilatérales". We thank the reviewers for their comments.

References

[1] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 585–599, 2016.

[2] W. E. Walsh and M. P. Wellman, "A market protocol for decentralized task allocation," in *ICMAS*, 1998, pp. 325–332.

[3] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artif. Intell.*, vol. 101, no. 1-2, pp. 165–200, 1998.

[4] J. Turner, Q. Meng, G. Schaefer, and A. Soltoggio, "Distributed strategy adaptation with a prediction function in multi-agent task allocation," in *Proc. of AAMAS*, 2018, pp. 739–747.

[5] Q. Baert, A.-C. Caron, M. Morge, J.-C. Routier, and K. Stathis, "Adaptive Multi-agent System for Situated Task Allocation," in *Proc. of AAMAS*, 2019, pp. 1790–1792.

[6] Q. Baert, A.-C. Caron, M. Morge, and J.-C. Routier, "Fair multi-agent task allocation for large datasets analysis," *Knowledge and Information Systems*, vol. 54, no. 3, pp. 591–615, Mar 2018.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Symposium on Operating Systems Design and Implementation*, 2004, pp. 137–150.

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

[9] Y. Kwon, K. Ren, M. Balazinska, and B. Howe, "Managing skew in Hadoop." *IEEE Data Eng. Bull.*, vol. 36, no. 1, pp. 24–33, 2013.

[10] B. Chen, C. N. Potts, and G. J. Woeginger, *Handbook of combinatorial optimization*. Springer, 1998, ch. A review of machine scheduling: Complexity, algorithms and approximability, pp. 1493–1641.

[11] E. Horowitz and S. Sahni, "Exact and approximate algorithms for scheduling nonidentical processors," *Journal of the ACM*, vol. 23, no. 2, pp. 317–327, 1976.

[12] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, 1977.

[13] A. M. A. Hariri and N. Potts, Chris, "Heuristics for scheduling unrelated parallel machines," *Computers & operations research*, vol. 18, no. 3, pp. 323–331, 1991.

[14] S. Martello, F. Soumis, and P. Toth, "Exact and approximation algorithms for makespan minimization on unrelated parallel machines," *Discrete applied mathematics*, vol. 75, no. 2, pp. 169–188, 1997.

[15] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical programming*, vol. 46, no. 1-3, pp. 259–271, 1990.

[16] B. An, V. Lesser, D. Irwin, and M. Zink, "Automated negotiation with decommitment for dynamic resource allocation in cloud computing," in *Proc. of AAMAS*, 2010, pp. 981–988.

[17] G. Attiya and Y. Hamam, "Task allocation for maximizing reliability of distributed systems: A simulated annealing approach," *Journal of parallel and Distributed Computing*, vol. 66, no. 10, pp. 1259–1266, 2006.

[18] P.-Y. Yin, S.-S. Yu, P.-P. Wang, and Y.-T. Wang, "Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization," *Journal of Systems and Software*, vol. 80, no. 5, pp. 724–735, 2007.

[19] Q.-M. Kang, H. He, H.-M. Song, and R. Deng, "Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization," *Journal of Systems and Software*, vol. 83, no. 11, pp. 2165–2174, 2010.

[20] S. K. Garg, S. Venugopal, J. Broberg, and R. Buyya, "Double auction-inspired meta-scheduling of parallel applications on global grids," *Journal of Parallel and Distributed Computing*, vol. 73, no. 4, pp. 450–464, 2013.

[21] A. Damamme, A. Beynier, Y. Chevaleyre, and N. Maudet, "The Power of Swap Deals in Distributed Resource Allocation," in *Proc. of AAMAS*, 2015, pp. 625–633.

[22] T. Sandholm, "Contract types for satisficing task allocation," in *Proceedings of the AAAI spring symposium: Satisficing models*, 1998, pp. 23–25.

[23] U. Endriss, N. Maudet, F. Sadri, and F. Toni, "Negotiating socially optimal allocations of resources," *JAIR*, vol. 25, no. 1, pp. 315–348, 2006.

[24] A. Nongaillard and P. Mathieu, "Reallocation problems in agent societies: A local mechanism to maximize social welfare," *Journal of Artificial Societies and Social Simulation*, vol. 14, no. 3, p. 21, 2011.

[25] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment," in *International Conference on Computer and Information Technology*. IEEE, 2010, pp. 2736–2743.

[26] M. Liroz-Gistau, R. Akbarinia, and P. Valduriez, "FP-Hadoop: efficient execution of parallel jobs over skewed data," *VLDB Endowment*, vol. 8, no. 12, pp. 1856–1859, 2015.

[27] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Transactions on computers*, vol. 29, no. 12, pp. 1104–1113, December 1980.

[28] B. Alrayes, Ö. Kafalı, and K. Stathis, "Concurrent bilateral negotiation for open e-markets: the CONAN strategy," *Knowledge and Information Systems*, vol. 56, no. 2, pp. 463–501, 2017.

[29] M. Schillo, C. Kray, and K. Fischer, "The eager bidder problem: a fundamental problem of dai and selected solutions," in *Proc. of AAMAS*, 2002, pp. 599–606.

[30] Q. Baert, A.-C. Caron, M. Morge, and J.-C. Routier, "Negotiation Strategy of Divisible Tasks for Large Dataset Processing," in *Proc. of EUMAS*, ser. LCNS, vol. 10767. Springer, 2017, pp. 370–384.

[31] ——. MAS4Data. https://github.com/cristal-smac/mas4data, visited 2019-07-01.

[32] Lightbend, Inc. Akka toolkit. https://akka.io, visited 2019-07-01.