An adaptive multi-agent system for task reallocation in a MapReduce job

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

An adaptive multi-agent system for task reallocation in a MapReduce job

Quentin Baert, Anne-Cécile Caron, Maxime Morge and Jean-Christophe Routier[1]

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

[1]Corresponding author. Tel: +33 (0)3 20 33 77 12 / Fax: +33 (0)3 28 77 85 37

Keywords: Artificial Intelligence, Multi-agent systems, Negotiation, MapReduce, BigData
2000 MSC: 00-01, 99-00

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. The adaptivity of these systems to various settings without configuration requiring user expertise, and their responsiveness to rapid changes need to be improved.

This paper is concerned with a class of practical applications where (a) the resources (e.g. data) required to successfully execute a task are distributed among nodes, (b) some of these nodes may encounter potential execution hazards, e.g. slowing down nodes or communication lags and (c) the number of tasks prevents from using a centralized approach to compute the allocation. As several resources are necessary to perform a task, any 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 their executions and should capitalize upon the way resources are distributed in the system.

In this class of practical applications, we consider here MapReduce [2] which is the most prominent distributed data processing model for tackling vast amount of data on commodity clusters. Tasks are divided into a set of map tasks and reduce tasks that are distributed on nodes. The task allocation among the reducers is *a priori* fixed by the partition function. For instance, the partition number is the hash value of the key modulus the number of partitions according to the default partition of the most popular implementation Hadoop [3]. Such a task allocation can be problematic. Firstly, several data skews in the MapReduce applications lead to an unbalanced workload during the reduce phase [4, 5].

Secondly, an unfair allocation can occur during the reduce phase due to the heterogeneous performance of nodes. Thirdly, the load-balancing can be challenged by some rapid performance variations due to exogenous reasons.

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 [6]. A multi-agent system is a decentralized system where multiple agents take local decisions based on their perceptions of the environment such that a solution to a complex problem can emerge from the interactions between simple individual behaviours [7]. Most of the existing works adopting the market-based approach [8, 9, 10] 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 considered in this paper. 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, as required by our practical application, that a task can be performed by any single agent without preemption and precedence order. A task is indivisible, with no deadline and not shareable i.e. a task belongs to only one agent at a time.

In this paper, we formalize the multi-agent situated task allocation problem. We propose a dynamic and on-going task reallocation process which takes place concurrently with the task execution and so the distributed system is adaptive to disruptive phenomena (e.g. slowing down nodes). When agents locally identify opportunities within a current unbalanced allocation, they trigger concurrent and one-to-many negotiations to reallocate some tasks. 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. This paper significantly

extends our previous works. Contrary to [11], we do not assume that each task has an intrinsic cost (e.g. the number of data to process) but the location of resources is taken into account. Beyond [12], we introduce here a multi-auction process which allows an agent to bid in several concurrent negotiations in order to improve the responsiveness of the system. Additionally, we present a vast panel of experiments in order to empirically evaluate (a) the adaptivity of our multi-agent system in an heterogeneous environment, (b) the responsiveness of the multi-agent system due to the multi-auction process, and (c) the adequacy of the agent strategy with respect to the computing infrastructure.

Specifically, our contributions are as follows:

- We formalize the multi-agent situated task allocation problem where tasks have different costs for different agents due to the resource locality.
- We design a multi-agent version of the MapReduce pattern in a distributed system setting which solves the partitionning data skew problem. The task reallocation process based on concurrent negotiations between agents occurs all along the MapReduce job processing to cope with a continuously evolving environment.
- We conduct extensive experiments on real-world datasets. The experimental results show that our method improves the runtime with a negligible computational overhead and it mitigates the heterogeneity of the computational environment.

The paper is structured as follows. Section 2 overviews relevant related works. Section 3 defines the socially rational task delegation considered by the agents in order to locally improve the task allocation. Section 4 sketches the negotiation process which is concurrent with task consumptions. Section 5 specifies the strategies, i.e. how agents choose which task to perform/negotiate. Our practical application and empirical evaluation are described in Section 6. Finally, Section 7 summarizes our contribution and outlines our future work.

2. Related work

The context of this paper is a single MapReduce job. In this context the problem of task scheduling consists of assigning map and reduce tasks as suggested by Selvitopi et al. in [13]. This problem should not be confused with that of job scheduling, where one considers the allocation and the usage of the resources in case of multiple MapReduce jobs as discussed by Banerjee and Hecker in [14]. Several common data skews in the MapReduce applications are identified in [4, 5] which lead to an unbalanced workload during the data processing. In this paper, we focus on the partitioning skew leading to an unbalanced allocation among the nodes where the reduce phase is slowed down since it is penalized by the most loaded reducer. This data skew is tackled by [15, 16] using parametrization based on prior knowledge about the data and the distributed computing environment. Every reduce task is scheduled in [17] according to the data skew and the data locality without modifing the partition. In this paper, we address the partitionning data skew 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 and the processing, (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.

We provide here a comparison of our work with the most significant existing methods for task allocation and load-balancing which is summarized in Table 1. This analysis grid classifies these works according to the crucial aspects which are requested by our practical application: the deployment of the MapReduce design pattern for processing large datasets.

Classical scheduling problems have been the subject of extensive research producing offline schedulers for some simple models [31]. 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

	Decentralization	Adaptation	Cooperation	Distribution	Objectives
Ibarra and Kim [18]	-	-	-	-	Makespan minimization
Lenstra et al. [19]	-	-	-	-	Makespan minimization
Hariri and Potts [20]	-	-	-	-	Makespan minimization
Martello et al. [21]	-	-	-	-	Makespan minimization
Jiang [22]	✓	✓	-	✓	Response time
Selvitopi et al. [13]	✓	-	-	✓	Makespan minimization
Di and Wang [23]	✓	✓	✓	✓	Throughput
Turner et al. [10]	✓	✓	✓	-	Throughput
Schaerf et al. [24]	✓	✓	✓	✓	Throughput
Jiang and Jiang [25]	✓	✓	✓	✓	Throughput
Jiang and Li [26]	✓	✓	✓	✓	Response time minimization
Jiang and Zhichuan [27]	✓	✓	✓	✓	Response time minimization
Walsh and Wellman [8]	✓	✓	-	✓	An allocation
Kraus et al. [28]	✓	✓	-	-	An allocation
An et al. [29]	✓	✓	-	✓	Response time minimization
Penmatsa and Chronopoulos [30]	✓	-	-	-	Response time minimization
Shehory and Kraus [9]	✓	✓	✓	-	Makespan minimization
MAS4Data	✓	✓	✓	✓	Makespan minimization

Table 1: Analysis grid of related works according to the main aspects.

$R||C_{max}$, is NP-hard [32]. Pseudo-polynomial algorithms developed for this problem include: the earliest completion time heuristic (ECT) proposed by Ibarra and Kim in [18], two-phase heuristics based on linear programming suggested by Lenstra et al. in [19], the local search heuristics proposed by Hariri and Potts in [20], and the branch and bound algorithm applied by Martello et al. in [21]. These centralized algorithms and the most recent ones [33], even the ECT heuristic which is an approximation algorithm giving acceptable results with very small computational requirements, cannot be applied to our scenario with a large number of tasks (e.g. $100,000$ keys in Sec. 6). The strategies presented in Sec. 5 are decentralized local search heuristics.

Multi-agent scheduling [6] has received significant attention for load-balancing problems in distributed systems, but it is different from the classical scheduling problems 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. Instead, task allocation can be negotiated by agents representing the nodes. For instance, Jiang et al. propose in [22] a negotiation reputation-based allo-

cation mechanism for load-balancing in order to reduce the resource access time and so the responding time.

- **Adaptation:** classical scheduling problems are static. The inaccurate estimation of tasks execution time and the disruptive phenomena (task consumption, slowing down nodes, etc.), may require major modifications in the existing allocation to stay optimal. Di and Wang introduce in [23] a self-adaptive system to implement a high adaptable task allocation to a dynamic environment. Turner et al. combine in [10] supervised classification learning with an internal decision-making process for task assignments. Schaerf et al. investigate in [24] the adaptive behaviour of agents for efficient load-balancing using multi-agent reinforcement learning. These methods cannot be used for the class of practical applications we are concerned since neither generalizable predictive patterns nor prior model of the data/environment are available.

The enormous amount of varying related studies (see [1] for a recent survey) mainly distinguish themselves through the following aspects:

- **Objectives:** the makespan minimization is the most widely applied optimization objective for task allocation. Selvitopi et al. [13] propose a task scheduling to minimize the makespan however the allocation is made once for all before the reduce phase while our method challenges the allocation all along the process. Minimizing the total or mean response time of all tasks means minimizing the waiting time of tasks. The throughput measures the number of tasks completed per time unit. The reliability measures the probability that the tasks can be successfully executed. Attiya and Hamam distinguish in [34] the node-related reliability, i.e. the reliability of resources/computation and the path-related reliability, i.e. the reliability of communications.

- **Distribution:** some resources are placed at the nodes and can be accessed and shared to execute the tasks. For this purpose, the location of agents

and the accessibility of required resources should be considered which is not the case of most of the task allocation algorithms. In [25], Jiang and Jiang consider that the number of allocated tasks on a node is proportional to its own resources and the resources of its interacting nodes. Jiang and Li propose in [26] a locality-sensitive resource allocation model which takes into account the distance between the nodes and the locality of resources. Jiang and Zhichuan present in [27] a task allocation mechanism based on the contextual resources: if a node has richer experiences of executing tasks, the node may have higher access to the resources. In this paper, the nodes are equidistant from each other in a fully connected physical and social network. The efficiency of a task allocation depends both on the proportion of resources (data) which are local and their intrinsic talent (CPU).

- **Cooperation:** Garg et al. distinguish in [35] the meta-schedulers which optimize system-centric metrics and the most recent ones focusing on user-centric metrics. Most of the latter adopts the market-based approach: they model the load-balancing problem as a non-cooperative game. Walsh and Wellman present in [8] a task allocation protocol among agents which acquire and provide goods on behalf of consumers or producers. Kraus et al. consider in [28] that, even if each agent tries to maximize its benefits, they need to cooperate to perform tasks. An et al. propose in [29] a distributed negotiation mechanism where selfish agents negotiate over resources both a contract price and a decommitment penalty. Penmatsa and Chronopoulos formulate in [30] the load balancing problem as a non-cooperative game among the users who try to minimize the expected response time of their own tasks. Fewer works are based on cooperative agents which negotiate to address system-centric metrics such as the global runtime we consider in this paper. For instance, Shehory and Kraus consider in [9] that task assignments to groups of agents as necessary since tasks cannot be performed by a single reliable agent. By contrast, we assume here 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 with no deadline.

3. 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* $MASTA = \langle Node, \mathcal{A}, \mathcal{T}, l, d, c \rangle$ *such that:*

- $Node = \{node_1, \ldots, node_k\}$ *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 are cheaper when the required resources are more local ones:*

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

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.

In the following of this section, 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 and 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:*

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

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

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

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

The makespan of P is defined as:

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

Let us consider the following walk-through example.

Example 1 (Task allocation, workload and makespan). *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 $l(1) = node_1$ and $l(2) = node_2$. The locations of the tasks and their costs are represented in Table 2. Let*

	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 2: Locations and costs of tasks

Pmks and P be two task allocations 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 $w_1(Pmks) = 33$ and $w_2(Pmks) = 35$ and so $C_{max}(Pmks) = 35$. As shown at top of Figure 1, it is not the case of P since $w_1(P) = 46$, $w_2(P) = 27$ and so $C_{max}(P) = 46$.

The agents modify the task allocation by negotiating task delegations. The delegation δ of the task τ from agent i to agent j aims at improving the

makespan, i.e the load-balancing between the two agents. The following definition formalizes the required conditions for an effective task delegation.

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:*

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

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

The delegation is socially rational iff:

$$w_j(P) + c_j(\tau) < w_i(P) \tag{8}$$

Since a socially rational task delegation δ strictly decreases the local makespan between the two agents, it does not increase the global makespan ($C_{max}\big(\delta(P)\big) \leq C_{max}(P)$).

We can now denote $\Gamma_i(P)$ the set of socially rational task delegations that an agent i can trigger:

$$\Gamma_i(P) = \{\tau \in P(i) \mid \exists j \in \mathcal{A} \setminus \{j\}, w_j(P) + c_j(\tau) < w_i(P)\} \tag{9}$$

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

Example 2 (Socially rational task delegation). *Let δ_1 be the delegation of the task τ_6 from agent 1 to agent 2 in the example 1 which leads to the task allocation $P' = \delta_1(P)$. Since $w_1(P') = 38$ and $w_2(P') = 40$, δ_1 improves the makespan: $C_{max}(P') = 40 < C_{max}(P) = 46$. 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. Figure 1 depicts the workloads of the two agents after the delegations δ_1 and δ_2.

It is worth noticing that the load-balancing process which consists of a sequence of socially rational delegations is finite.

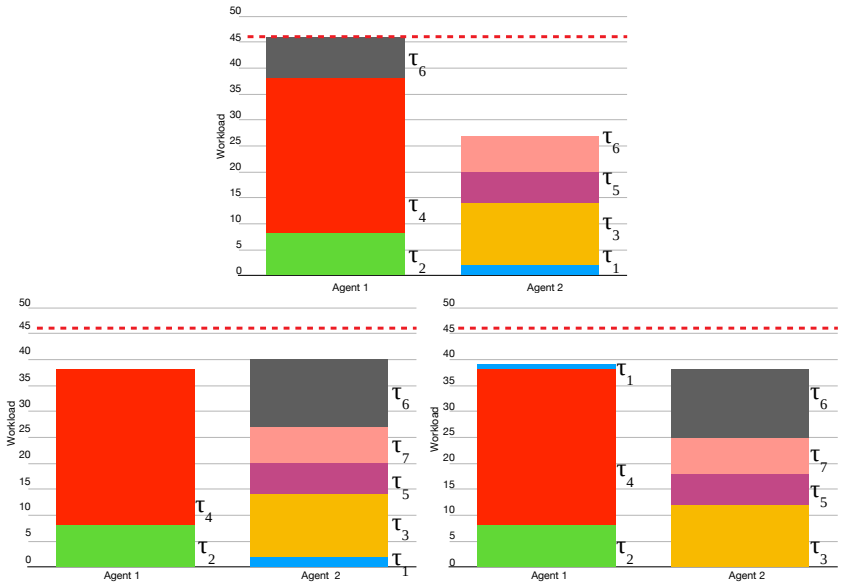


Figure 1: The workloads of the agents 1 and 2 for the allocations P (at top), P' (at bottom left) and P'' (at bottom right) in our walk-through example.

Property 1 (Load-balancing process). *An unstable task allocation can always lead to a stable one using a finite number of socially rational task delegations.*

Proof 1 (Load-balancing process). *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,*

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

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

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.

4. Negotiation process

We sketch here the repeated negotiations process which is concurrent with task execution. When a task is consumed, it will be removed from the set of tasks, and so the multi-agent system aims at minimizing the makespan of the current allocation for this new MASTA problem.

The execution of a task is a disruptive event which modifies the MASTA problem and the task allocation. Formally,

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 the 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:*

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

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

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

A sequence of task consumptions removes all the tasks from the initial allocation until a final empty one, denoted $\perp$. Obviously, a task consumption may decrease the makespan.

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 [36], includes three decision steps: (a) the choice of the task to negotiate by the strategy of the initiator described in Section 5, (b) the refusals/bids from the peers which check the social rationality of the task delegation, and (c) the selection of the winning bid by the initiator. We consider here that the initiator selects the bidder with the smallest workload in order to decrease the makespan. 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 the 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,

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

If $\Gamma_i^{\mathcal{B}}(P) = \emptyset$, then agent does not initiate negotiations. The computation of the local makespan by the initiator of a negotiation is also based on its belief base, possibly inaccurate. This is the price to pay for decentralization. However, an agent informs its peers about its workload when it is triggered for the first time, within the negotiation messages, and after each task execution. Therefore, the belief base of the peers is updated. It follows that a successful negotiation can only reach a socially rational task delegation, and so tends to improve the makespan. When an agent, according to its belief base, identifies opportunities within a current unbalanced allocation, it initiates negotiations. These local decisions promote the adaptivity of the multi-agent system.

Concurrent consumptions and delegations. Task delegations and task consumptions are concurrent and complementary operations since a task removal may be an opportunity for new socially rational task delegations. Figure 2 represents the influence of these operations over the path from the initial allocation P_0 until the final one $\perp$. 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).

Multi-auction. First of all, even if an agent which is involved in a negotiation as a bidder cannot initiate another negotiation and conversely, several negotiations involving different groups of agents may concurrently occur. Additionally,

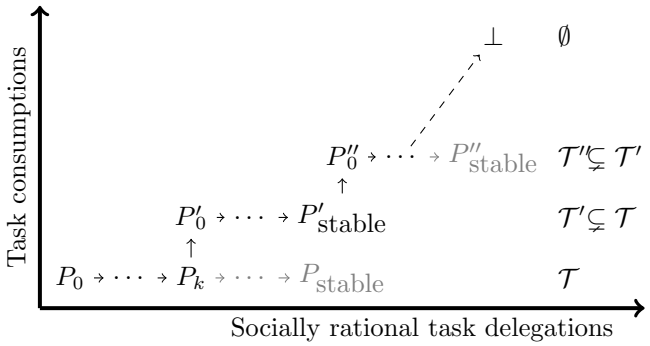


Figure 2: Concurrent task consumptions (vertical edges) and task delegations (horizontal edges).

we introduce here a multi-auction process which allows agent to bid in several concurrent negotiations as in [37] in order to improve the responsiveness of the system. In this way, as stated by our empirical results in Section 6, the gap between the most loaded reducer and the least loaded one is filled faster and so the load-balancing process is faster.

In order to tackle the eager bidder problem [38], we adopt a conservative approach which warrants that the task delegations are socially rational. For this purpose, a bidder computes an overhead.

Definition 5 (Overhead). *Let P be an allocation, $\mathcal{D} \subset \mathcal{T}$ the tasks which are currently negotiated and $\mathcal{D}_j$ the set of pending tasks for which the agent j has made proposal. The overhead of the agent j is:*

$$v_j(\mathcal{D}) = \Sigma_{\tau \in \mathcal{D}_j} c_j(\tau) \tag{17}$$

The potential workload of the agent j, which represents its workload if it wins all the auctions in which it is involved ($w_j(P) + v_j(\mathcal{D})$), allows a bidder not to be too optimistic and to make proposals which only lead to socially rational task delegations. In order to evaluate the delegation of the task τ from the agent i, the bidder j adopts the following strategy:

- either $w_j(P) + c_j(\tau) \geq w_i(P)$ so the bidder declines the delegation which is not socially rational;

- or $w_j(P)+c_j(\tau) < w_i(P) \leq w_j(P)+v_j(\mathcal{D})+c_j(\tau)$ so the bidder postpones the evaluation of the delegation which depends on the outcomes of the pending negotiations;

- or $w_j(P)+v_j(\mathcal{D})+c_j(\tau) < w_i(P)$ so the bidder makes a proposal since the task delegation is socially rational whatever the outcomes of the pending negotiations are. Then,

$$v_j(\mathcal{D} \cup \{\tau\}) = v_j(\mathcal{D}) + c_j(\tau) \tag{18}$$

315 Moreover, the bidder informs the initiator about its potential workload.

In the latter case, when the negotiation closes:

- either the bidder is selected ($P'_j = P_j \cup \{\tau\}$) and its workload is updated $w_j(P') = w_j(P) + c_j(\tau)$;

- or it is not the case ($P'_j = P_j$), its workload remains the same $w_j(P') =$
320 $w_j(P)$.

In both cases, the overhead is updated:

$$v_j(\mathcal{D}) = v_j(\mathcal{D} \cup \{\tau\}) - c_j(\tau) \tag{19}$$

Finally, the pending delegations are re-evaluated.

Example 3 (Multi-auction). *Let P be an allocation among the set of agents $\mathcal{A} = \{1, 2, 3, 4, 5, 6\}$ such that the workloads are:*

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

325 *We focus here on the overhead of the agent 1 and it influence over the negotiations. We assume the agents 2, 3, 4, 5 and 6 ask for the delegation of the tasks τ_2, τ_3, τ_4, τ_5 and τ_6 respectively. The cost of these tasks for the agent 1 are:*

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

Figure 3 illustrates the evolution of the overhead for the agent 1 *during its interactions with its peers. The agent* 1 *can make a proposal to the agents* 2 *and* 3 *(messages* 2 *and* 4*). However, the delegation requested by the agent* 4 *(message* 5*), which is not socially rational, is declined (message* 6*). The agent* 1 *must postpone the evaluation of the delegations from the agents* 5 *and* 6 *(messages* 7 *and* 8*) which depends on the previous pending negotiations. When the delegation of the task* τ_2 *is confirmed (message* 10*), the agent* 1 *can decline the delegation of* τ_5 *(message* 11*) but the social rationality of the delegation of the task* τ_6 *is neither excluded nor confirmed. Finally, when the delegation of the task* τ_3 *is rejected (message* 12*), the agent* 1 *can make a proposal about* τ_6 *(message* 12*).*

5. Strategies

Since task delegations and task consumptions are concurrent, the strategy of an agent must select the next task to perform/delegate.

Definition 6 (Strategy). *Let* P *be an allocation, the strategy of the agent* i *is the couple* $(perform_i, negotiate_i)$ *where:*

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

It is worth noticing that the delegation which is selected by the strategy must be a potential socially rational one. In the following, we propose two strategies. While the local agnostic strategy is only based on the cost function, the location-aware one takes into account the location of the required resources.

Local agnostic strategy. By adopting the principle "consume small, delegate big", an agent performs the smallest task in its bundle and negotiate the largest one which may lead to a potential socially rational delegation. For this purpose,

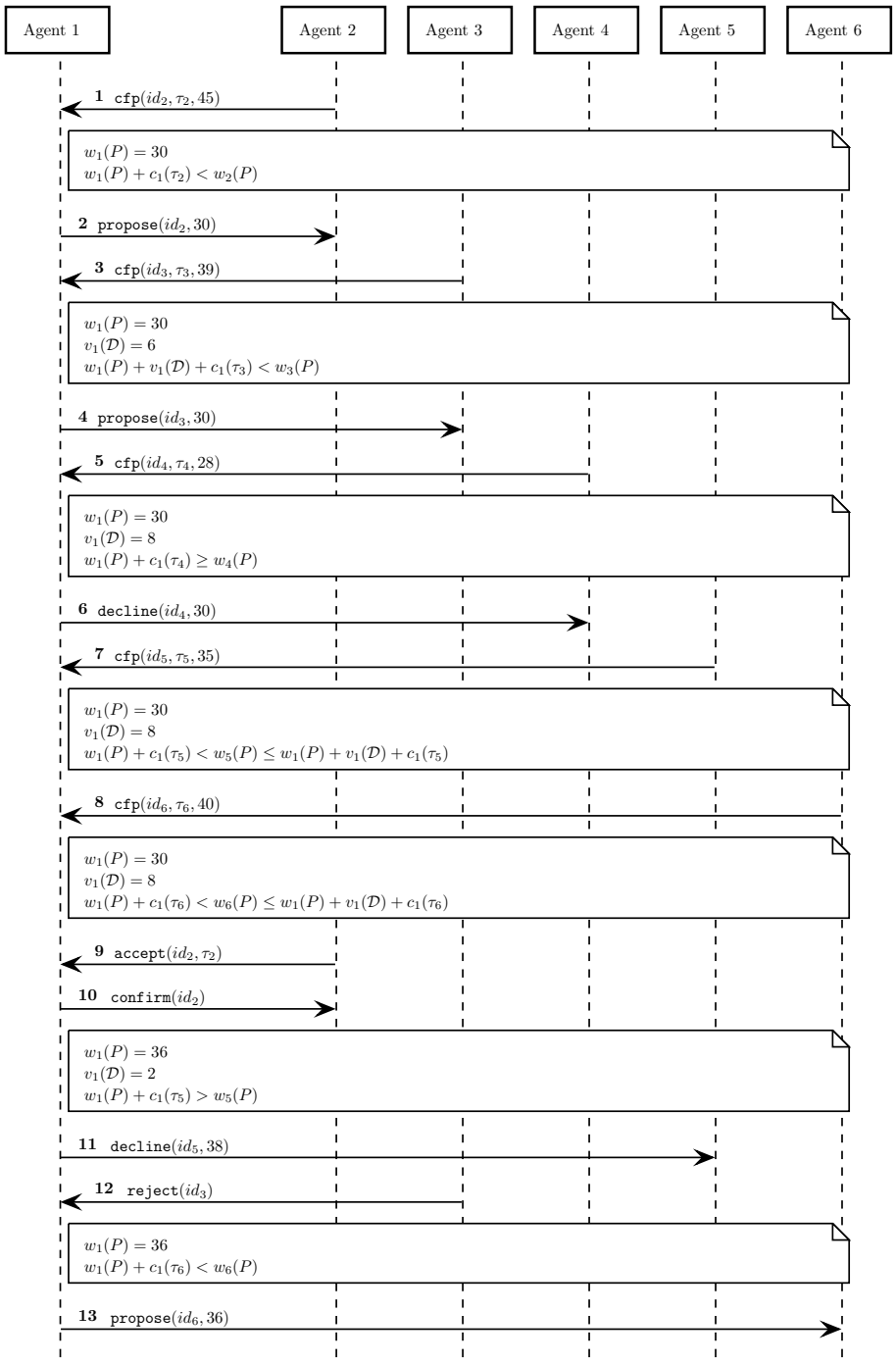


Figure 3: The agent 1 bids in several concurrent negotiations as described in Example 3.

this strategy only requires that the agent sorts the tasks according to their costs. We can notice that the consideration of the resource fetching time by this strategy is implicit since an agent should perform first the local tasks which may cost more for its peers and delegate first the distant tasks which may cost less for its peers. The following strategy makes this principle explicit.

Location-aware strategy. 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. This strategy is built on a data structure, called local-aware bundle.

Firstly, the local availability ratio measures the locality of tasks:

Definition 7 (Local availability ratio). *The local availability ratio of the agent i for the task τ is defined as:*

$$o_i(\tau) = \frac{d_i(\tau)}{d_\tau} \tag{20}$$

The maximum local availability ratio for the task τ is:

$$\hat{o}(\tau) = max_{i \in \mathcal{A}}\{o_i(\tau)\} \tag{21}$$

The local availability ratio of an agent i for a task is the ratio between the number of local resources and the total number of resources for this task.

Secondly, the **local-aware bundle** of agent i, which is depicted in Figure 4, is divided in three subbundles in accordance with the local availability ratios of the agent for the tasks:

1. *The maximum local bundle* 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. The tasks are sorted by decreasing order of cost (cf. left of Figure 4);
2. *The intermediate local bundle* contains the tasks which are partially local. The tasks are sorted by 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 Figure 4);

3. *The distant bundle* contains the tasks which are distant. The tasks are sorted by increasing order of cost (cf. right of Figure 4).

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 potential socially rational delegation.

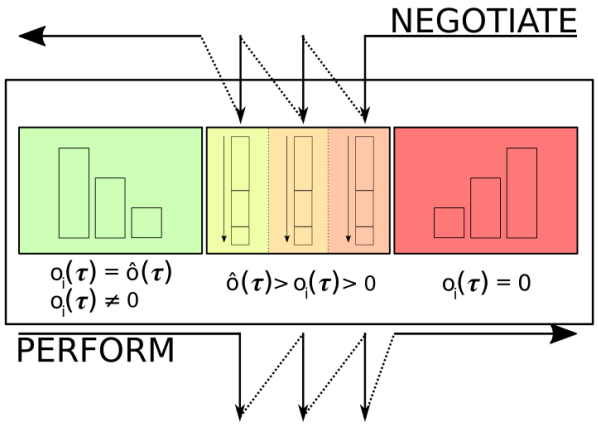


Figure 4: The local-aware bundle contains the maximum local bundle (at left), the intermediate local bundle (at center) and the distant bundle (at right). The rectangle size represents the cost of the task. The arrows depict the order in which an agent looks for a task to perform/negotiate.

6. Results and discussion

After introducing our practical application, we describe our prototype and we discuss our empirical results.

6.1. Practical application

We consider as a practical application the distributed deployment of the MapReduce design pattern in order to process large datasets on a cluster [2], as with Hadoop [3]. 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. Each data chunk is located on the same node as the mapper which has generated it. During the reduce phase, nodes process in parallel the keys and their individual lists of values.

The reduce phase of a MapReduce job can be formalized by a MASTA problem $\langle Node, \mathcal{A}, \mathcal{T}, l, d, c\rangle$ (cf. Definition 1) where: $Node$ is the set of nodes in the cluster, $\mathcal{A}$ is the set of reducer agents, $\mathcal{T}$ is a set of reduce tasks, l captures the fact that we put one reducer per node in our experiments and d considers the data location. In conformance with Equation 1, we specify the cost of a task τ for an agent i as follows:

$$c_i(\tau) = \sum_{\rho \text{ is a chunk for the task } \tau} c_i(\rho), \text{ with } c_i(\rho) = \begin{cases} |\rho| \text{ if } \rho \text{ is local for } i \\ \kappa \times |\rho| \text{ otherwise} \end{cases} \quad (22)$$

where $|\rho|$ denotes the number of values for the chunk ρ and κ captures the resource fetching time. We empirically set up $\kappa = 2$ for a cluster and $\kappa = 10$ when we use a network of computers. Our experiments show that the cost function does not need to be carefully tuned since the adaptivity of our dynamic task reallocation process allows to mitigate an inaccurate cost function.

6.2. Implementation

We have developed a multi-agent version of the MapReduce pattern in a distributed system setting using the MAS4Data testbed [39]. MAS4Data is implemented in Akka [40] 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 the message transmission delay is arbitrary but not negligible and that messages may be lost. These are the reasons why we have included acknowledgments and deadline mechanisms in the interaction protocol. In order to decrease the complexity related to the design of a reducer agent, we have adopted a modular agent architecture that allows the concurrency of the negotiations and the tasks performance, as well as the separation between communicative and decision-making behaviours.

6.3. Empirical results

This section starts by explaining the experimental setting in details, which includes the choice of the metrics, the datasets and the jobs. Then, we report on the experimental results.

Metrics. In order to evaluate the runtime and the fairness of our experiments we have introduced in our previous works [11, 12] the following metrics:

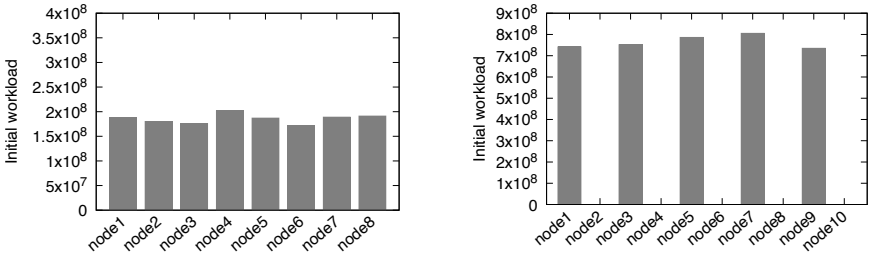
- The **contribution** of a reducer is the sum of the costs of the tasks it has performed;
- The **contribution fairness** is the ratio between the minimum and the maximum contributions of the agents;
- The **runtime** of the reduce phase is the runtime of the reducer which finished last;
- The **time fairness** is the ratio between the runtime of the slowest reducer and the runtime of the fastest one.

While the contribution of a reducer corresponds to its *ex-post* workload (see Definition 2), the runtime effectively measures the makespan. The closer to 1 the contribution fairness and time fairness are, the fairer is the allocation. In other words, we want to keep every agent as busy as all the other ones, thus sharing as much as possible the workload of the application.

Setup. We consider here two different hardware configurations: (a) a cluster with 10 blades, each having 10 CPUs with 512Go RAM; (b) a network of 16 PCs with 4 cores Intel(R) i7 and 16GB RAM each. Since we obtain similar results with the second configuration, we present here the experiments on the cluster with an exception for the last experiment which compares the two configurations. We use two distinct real-world datasets. The first dataset (2.4 Gb) contains $100,480,507$ ratings that $480,189$ users gave to $17,770$ movies [41] [dataset]. The job, called `RecByMov`, counts the number of rankings per movie. The second

dataset (977 Mb) contains 3,963,480 weather records (station id, timestamp, temperature, rainfall, etc.) from 62 stations taken during the last 20 years [42] [dataset]. The job, called `RecByTempSta`, counts the number of records per half degree of temperature and per station. In order to increase the volume of data without slowing down the mapping phase (which is out of the scope of this paper) and without changing the data distribution, we replicate k times the number of values for each key: $k = 200$ for the job `RecByTempSta` and $k = 20$ for the job `RecByMov`. Whatever the job and the datasets are, we use as many mappers as reducers and each experiment is run 30 times. Due to the observable nondeterminism of distributed execution, we comment distinctive runtimes.

Figure 5 shows the initial workloads for these two jobs with the default Hadoop partition function, i.e. the hashcode of the key modulo the number of reducers. While the first partition is well balanced, the second one is intentionally caricatural[2] since half of the reducers have no task to perform. The latter highlights the partitioning skew [4, 5] since half of the reducers execute no task.



(a) The job `RecByTempSta` on 8 nodes

(b) The job `RecByMov` on 10 nodes

Figure 5: The allocation for our two jobs with the classical Hadoop

Empirical results . Firstly, our experiments aims at validating our approach. We show that: (a) the negotiation improves the makespan and so the runtime with a small overhead cost for the negotiation, and (b) our multi-agent system is adaptive to performance variations and heterogeneous environments. We adopt

[2]For this purpose, we consider `Double` for the type of the keys, i.e. the movie ids.

by default the local agnostic strategy and the multi-auction process. Secondly, we evaluate these choices.

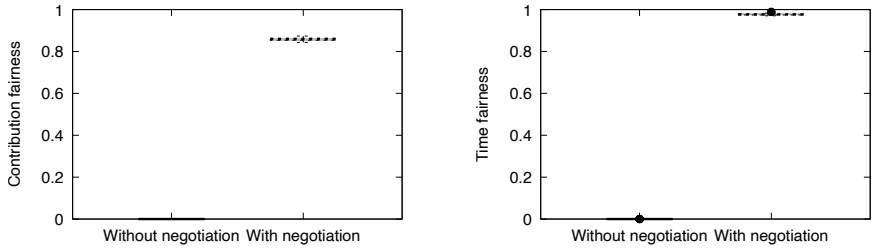
Negotiation improves the runtime. Negotiation is beneficial whether or not the initial allocation is fair. In order to validate this hypothesis, we compare the contribution fairness, the time fairness and the runtime of jobs with/without negotiations. Note that, throughout the paper, whenever we use the term "without negotiation" we refer to results obtained "with the classical Hadoop".

Empirical result 1. *The negotiation improves the runtime due to the load-balancing of the reducers contributions.*

Figure 6 shows that a job can strongly benefit from the negotiations. Starting from an initial unfair allocation (cf. Figure 5b), we execute 30 times the job `RecByMov` with 10 reducers on 10 homogeneous nodes. During the data processing, task delegations occur since the workloads are unbalanced. The most loaded reducers propose tasks which are accepted by the less loaded ones. Finally, the contributions are fairly distributed among the nodes and all the reducers terminate at the same time since the fairnesses are close to 1 (cf. Figures 6a and 6b). Therefore, the makespan (cf. Figure 6d) and so the runtime (cf. Figure 6c) are more than halved.

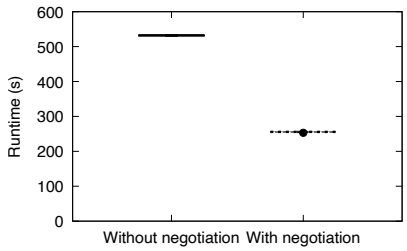
Empirical result 2. *When the initial allocation is fair, the overhead of the negotiation is negligible and it does not affect the runtime.*

Figure 7 shows that a job cannot be penalized by the negotiations. Starting from an initial fair allocation (cf. Figure 5a), we execute 30 times the job `RecByTempSta` with 8 reducers on 8 homogeneous nodes. Since the initial allocation is fair, it is not surprising that the contribution fairness (cf. Figure 7a) and the time fairness (cf. Figure 7b) are quite good even without negotiation, i.e. close to 0.85. However, they are still improved by negotiation since the makespan can be slightly decreased (cf. Figure 7d). Therefore, the runtime is about 7% faster with negotiation (cf. Figure 7c). It is worth noticing that no negotiation is triggered when the agents believe that the allocation is stable.

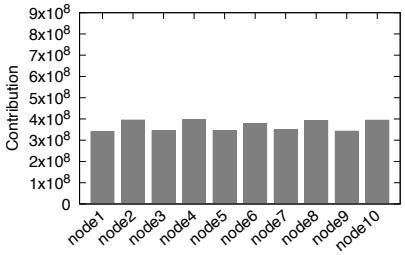


(a) The contribution fairness

(b) The time fairness

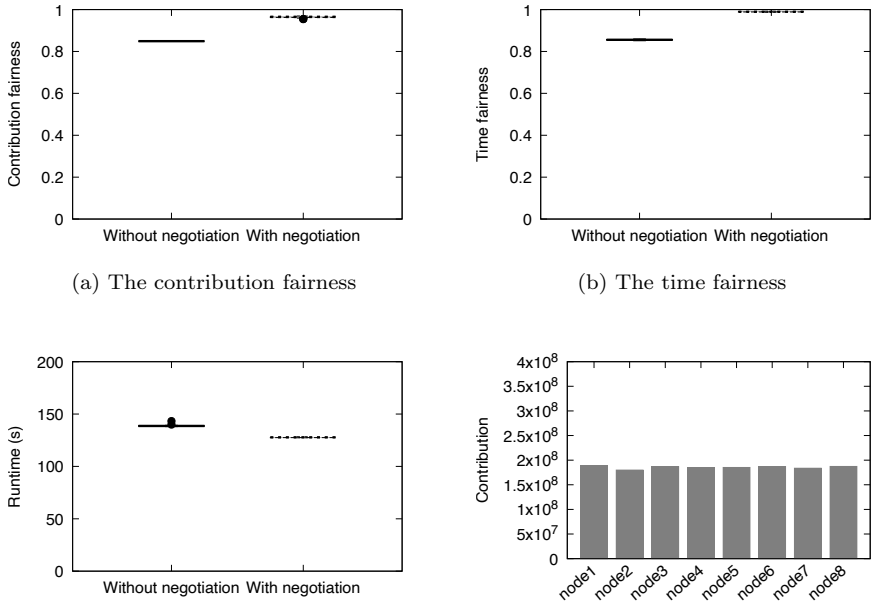


(c) The runtime



(d) The contributions

Figure 6: The median metrics and their standard deviations depicted in boxplots (a,b,c), and the contributions after negotiation for a distinctive execution of the job `RecByMov` (d).



(a) The contribution fairness

(b) The time fairness

(c) The runtime

(d) The contributions

Figure 7: The median metrics and their standard deviations depicted in boxplots (a,b,c) and the contributions after negotiation for a distinctive execution of the job `RecByTempSta` (d).

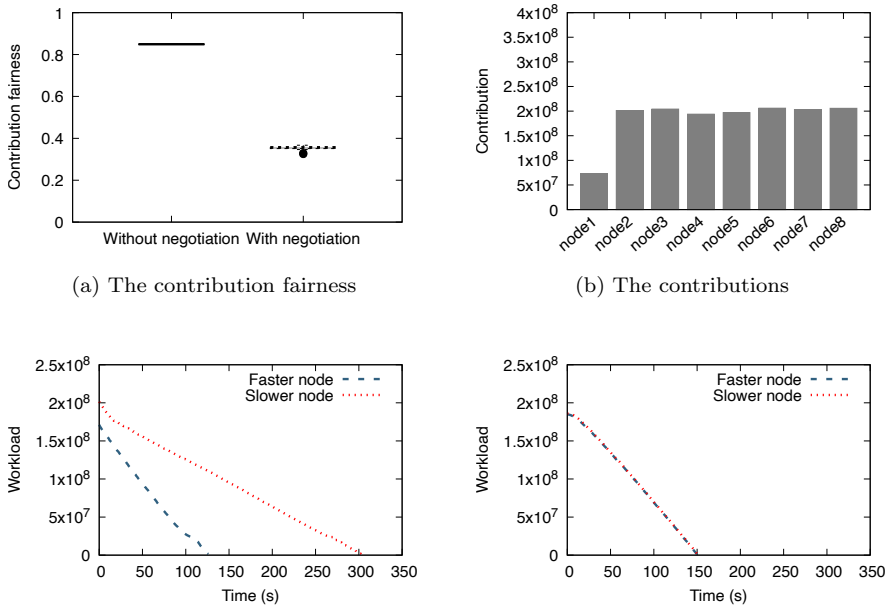
These two experiments validate that our adaptive dynamic process based on concurrent negotiations for tasks reallocation improves the runtime even if the initial allocation is fair. The first experiments demonstrates how MAS4Data tackles the partitioning skew. The second one highlights that the overhead of negotiation is negligible with respect to the benefit of the load-balancing.

Negotiation mitigates heterogeneity. The heterogeneity of a computational environment comes from a permanent non-uniformity of the processing capabilities of the nodes or a temporary one due to exogenous reasons (e.g. the slowdown of a node).

Empirical result 3. *The negotiation allows in an heterogeneous environment to reallocate the tasks to the fastest nodes in order to improve the runtime.*

Figure 8 shows that the multi-agent system adapts the allocation to runtime hazards. Starting from an initial fair allocation (cf. Figure 5a), we execute 30 times the job *RecByTempSta* with 8 reducers. Indeed, only one CPU is activated on each node and the first reducer cannot use more than 50% of the CPU time. Therefore, the seven other reducers run faster. The initial fair allocation is challenged by the slower reducer. Without negotiation (cf. Figure 8c), seven reducers terminate after 130 seconds while the slower reducer ends after 300 seconds and so the time fairness is low (around 0.43). By contrast, the negotiation allows the slower reducer to terminate after 150 seconds like the other ones (cf. Figure 8d). The time fairness is very close to 1 since some negotiations are triggered during the job processing for load-balancing. Thanks to this dynamic and continuous task reallocation process, the job is not penalized by the slower node and the job runs two times faster due to the negotiations. Finally, the contribution of the slower reducer is lower than the contributions of the other ones (cf. Figure 8b), and so the contribution fairness is low, approximately 0.35 (cf. Figure 8a).

This experiment shows that the negotiation process mitigates the impact of a runtime hazard which slows down one reducer. A reducer which is slowed down can delegate some tasks in order to execute the job as soon as possible.



(a) The contribution fairness

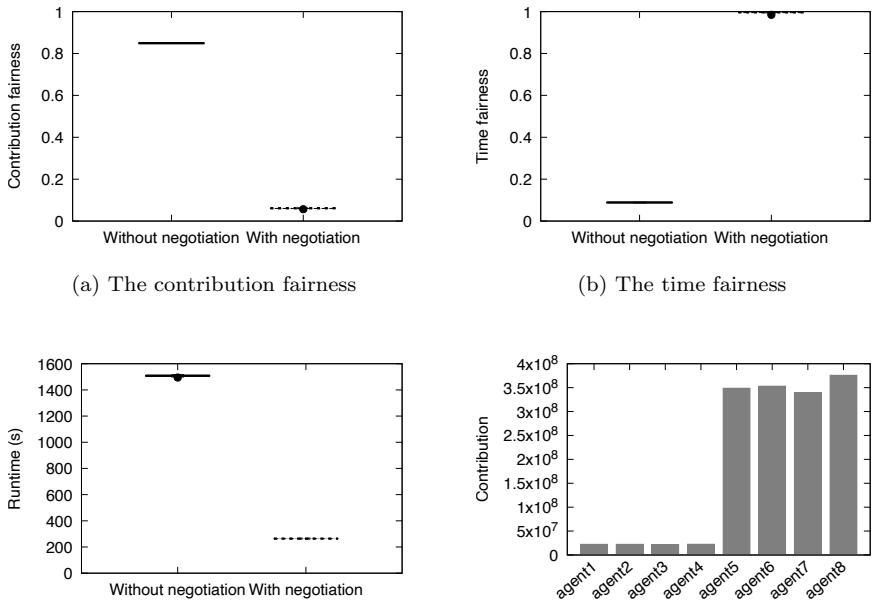
(b) The contributions

(c) Without negotiation

(d) With negotiation

Figure 8: The median contribution fairness and its standard deviations depicted in boxplots (a), the contributions (b) for a distinctive execution of the job `RecByTempSta` when a node is slowed down, the evolution of the workloads without negotiation (c) and with negotiation (d).

Figure 9 shows that the multi-agent system adapts the allocation to heterogeneous environment. Starting from an initial fair allocation (cf. Figure 5a), we execute 30 times the job *RecByTempSta* with 8 reducers. Indeed, only one CPU is activated on each node, four reducers run on the same node and four reducers run on separate nodes. As previously, the contribution fairness with negotiation is worst since the four slower reducers perform less task than the others. Once again, the dynamic and continuous task reallocation process allows the time fairness to reach approximately 1 and the job runs five times faster due to the negotiations. Finally, the contribution of the slower reducers is lower than the contributions of the others (cf. Figure 9d), and so the contribution fairness is low.



(a) The contribution fairness

(b) The time fairness

(c) The runtime

(d) The contributions

Figure 9: The median metrics and their standard deviations depicted in boxplots (a, b, c) and the contributions after negotiation for a distinctive execution of the job `RecByTempSta` in an heterogeneous environment (d).

These two experiments validate that a job running in an heterogeneous en-

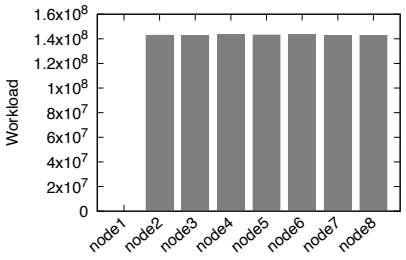
vironment benefits from the adaptivity of our multi-agent system. When a performance variation brings to an unbalanced allocation, our continuous and dynamic process detects it and triggers the reallocation of tasks toward the fastest nodes leading to the speedup of the runtime.

Single-auction versus multi-auction. We assume that our multi-auction process, which allows agents to bid in several concurrent negotiations, improves the responsiveness of the multi-agent system.

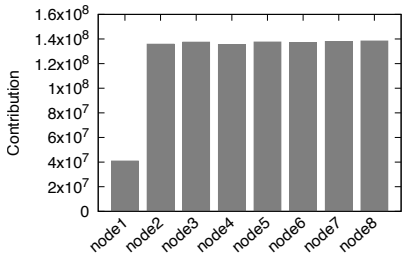
Empirical result 4. *The multi-auction process reaches faster stable allocation.*

Figure 10 shows that the multi-agent system adapts the allocation faster with a multi-auction process. We have generated a dataset (775 Mb) with $100,000,000$ lines such that the initial task allocation is unfair. The job, called `RecByKey`, counts the number of values per key. We use 8 reducers in an homogeneous environment where none task is assigned to the first reducer while $100,000$ tasks with $1,000$ values per task are assigned to the others reducers which have similar workloads (cf. Figure 10a). As in the previous experiments, the negotiation improves the load-balancing (cf. Figure 10b and 10c). However, the multi-auction process is more efficient. While the multi-auction process needs 50 seconds to reach a fair allocation (cf. Figure 10e), the workload of the first agent remains close to 0 during the whole reduce phase with the single-auction process (cf. Figure 10d). Actually, the tasks are delegated in the single-auction process to the first reducer one after the other and they are instantly performed. By contrast, this agent continuously bids in seven simultaneous auctions during the multi-auction process. Therefore, the data processing with a single-auction process ($\sim$ 275s) is around 10% slower than with a multi-auction one ($\sim$ 245s). Finally, the contribution fairness is around 0.3 with a single-auction while it is close to 0.85 with a multi-auctions process.

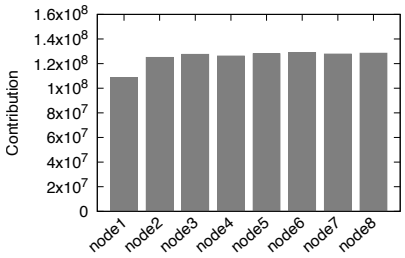
This experiment shows that the multi-auction process speeds up the load-balancing process and so improves the responsiveness of the multi-agent system.



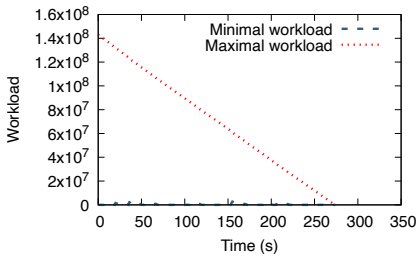
(a) The initial workloads



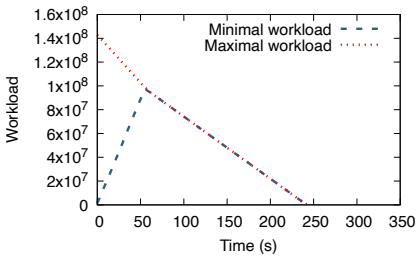
(b) The contributions with a single-auction



(c) The contributions with multi-auction



(d) The workloads with a single-auction



(e) The workloads with multi-auction

Figure 10: The initial workloads, the contributions and the workloads for a distinctive execution of the job `RecByKey` with single-auction (b and d) or multi-auction (c and e).

Adequacy of the strategy with respect to the infrastructure. We assume that the efficiency of the location-aware strategy depends on the resource fetching time.

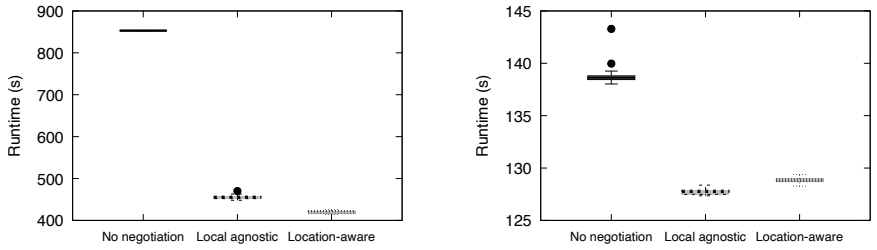
Empirical result 5. *The location-aware strategy improves the runtime when the extra cost for fetching resources is significant.*

Figure 11 shows that the efficiency of the negotiation strategy depends on the hardware configurations. We have generated a dataset (8 Gb) with $82,283$ keys such that the initial task allocation for the job `RecByKey` can be challenged. In order to evaluate the impact of the proximity between data resources and processing nodes on the runtime, most of the data required for a task are not located on the same node as the assigned reducer (See [12] of more details). With a network of computer, the location-aware strategy significantly improves the runtime with respect to the local agnostic strategy, around -7.6% (cf. Figure 11a). By contrast, the local agnostic strategy is more efficient within a cluster (cf. Figure 11b). Indeed, the cost of fetching distant resources from other nodes in a network of computers has a real impact and so the local execution of tasks speeds up the runtime. By contrast, this extra cost for task execution is low within a cluster. We experimentally measure it to be around 10%. Moreover, it is worth noticing that the locality is implicitly taken into account in the local agnostic strategy since the underlying cost function is defined such that the tasks are cheaper when the required resources are more local ones (cf. Equation 1).

Whatever the dataset and the job are, we observe that the location-aware strategy is the best strategy over a network of computers while the local agnostic strategy is more suitable on a cluster.

7. 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 to



(a) Network of computers (b) Computer cluster

Figure 11: The runtimes for a distinctive job on a network of computers (a) and on a cluster (b).

perform these tasks in order to minimize the makespan. In particular, we have applied our negotiation framework for load-balancing the reduce phase in the distributed MapReduce model in order to process large datasets.

Our prototype has been empirically evaluated. Our experiments show that MAS4Data is adaptive to the partitioning skew, an heterogeneous computing environment, and any potential execution hazards. This is due to the fact that the negotiation process improves the runtime due to the load-balancing of the reducers contributions. Some of our experiments suggest that future work should consider: (a) task swaps to improve the makespan of stable allocations and (b) task bundles in order to speedup the negotiation process. MAS4Data is scalable since it tackles a large number of tasks due to the local decisions of agents about the next task to delegate/perform. Moreover, the overhead of the negotiation is negligible with respect to the benefit of the load-balancing since the task reallocation is concurrent with the task consumption and no negotiation is triggered when the agents believe that the allocation is stable. Indeed, our method is not a scheduling algorithm which allocate the tasks once and for all but an ongoing distributed strategy attempting to repair a potential unbalanced partition.

From the user perspective, even if our adaptive and dynamic approach tackles the problem of performance drop, the fault tolerance can still be achieved

through data replication. It is worth noticing that MAS4Data does not require expertise for the parametrization due to its adaptivity. Even if the choice of the strategy depends on the hardware configuration, no other parameter needs to be carefully tuned such as the replication factor (by default the value 3 in HDFS [3]). A sensitivity analysis to study the influence of this parameter has been beyond the scope of this work, but it is certainly worth of further investigation.

Generally, future work must consider here the continuous arrival of complex jobs concurrently submitted by several users. Our study focuses on the reassignment of independent fine-grained tasks in a single job during their execution. In order to fill this granularity gap, the formal framework must be extended and the optimization objective to be considered should be the mean flowtime of several concurrent jobs i.e. the mean of the maximum completion times of dependent tasks in these jobs.

Grant

This work is supported by the CNRS Challenge Mastodons and the call ULille "Internationalisation Actions bilatérales". We thank the anonymous reviewers for their stimulating comments which help us to improve the paper.

References

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

[2] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, in: Proc. of the 9th Symposium on Operating Systems Design and Implementation, 2004, pp. 137–150.

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

[4] Y. Kwon, M. Balazinska, B. Howe, J. Rolia, Skewtune: mitigating skew in mapreduce applications, in: Proc. of the SIGMOD International Conference on Management of Data, ACM, 2012, pp. 25–36.

[5] Y. Kwon, K. Ren, M. Balazinska, B. Howe, Managing skew in Hadoop., IEEE Data Eng. Bull. 36 (1) (2013) 24–33.

[6] A. Agnetis, J. Billaut, S. Gawiejnowicz, D. Pacciarelli, A. Soukhal, Multi-agent Scheduling - Models and Algorithms, Springer, 2014.

[7] J. Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[8] W. E. Walsh, M. P. Wellman, A market protocol for decentralized task allocation, in: Proc. of the International Conference on Multi-Agent Systems (ICMAS), 1998, pp. 325–332.

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

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

[11] Q. Baert, A.-C. Caron, M. Morge, J.-C. Routier, Fair multi-agent task allocation for large datasets analysis, Knowledge and Information Systems 54 (3) (2018) 591–615.

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

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

[14] S. Banerjee, J. P. Hecker, A multi-agent system approach to load-balancing and resource allocation for distributed computing, in: First Complex Systems Digital Campus World E-Conference 2015, Springer International Publishing, 2017, pp. 41–54.

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

[16] M. Liroz-Gistau, R. Akbarinia, P. Valduriez, FP-Hadoop: efficient execution of parallel jobs over skewed data, VLDB Endowment 8 (12) (2015) 1856–1859.

[17] M. Hammoud, M. Rehman, M. Sakr, Center-of-Gravity Reduce Task Scheduling to Lower MapReduce Network Traffic, in: Proc. of the 5th International Conference on Cloud Computing (CLOUD), 2012, pp. 49–58.

[18] O. H. Ibarra, C. E. Kim, Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors, Journal of ACM 24 (2) (1977) 280–289.

[19] J. K. Lenstra, D. B. Shmoys, E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, Mathematical programming 46 (1-3) (1990) 259–271.

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

[21] S. Martello, F. Soumis, P. Toth, Exact and approximation algorithms for makespan minimization on unrelated parallel machines, Discrete applied mathematics 75 (2) (1997) 169–188.

[22] Y. Jiang, Y. Zhou, W. Wang, Task allocation for undependable multiagent systems in social networks, IEEE Transactions on Parallel and Distributed Systems 24 (8) (2012) 1671–1681.

[23] S. Di, C.-L. Wang, Decentralized proactive resource allocation for maximizing throughput of P2P Grid, Journal of Parallel and Distributed Computing 72 (2) (2012) 308–321.

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

[25] Y. Jiang, J. Jiang, Contextual resource negotiation-based task allocation and load balancing in complex software systems, IEEE Transactions on Parallel and Distributed Systems 20 (5) (2008) 641–653.

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

[27] Y. Jiang, Z. Huang, The rich get richer: Preferential attachment in the task allocation of cooperative networked multiagent systems with resource caching, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 42 (5) (2012) 1040–1052.

[28] S. Kraus, O. Shehory, G. Taase, Coalition formation with uncertain heterogeneous information, in: Proc. of 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2003, pp. 1–8.

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

[30] S. Penmatsa, A. T. Chronopoulos, Game-theoretic static load balancing for distributed systems, Journal of Parallel and Distributed Computing 71 (4) (2011) 537–555.

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

[32] E. Horowitz, S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, Journal of the ACM 23 (2) (1976) 317–327.

[33] E. Mokotoff, Parallel machine scheduling problems: A survey, Asia-Pacific Journal of Operational Research 18 (2) (2001) 193.

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

[35] S. K. Garg, S. Venugopal, J. Broberg, R. Buyya, Double auction-inspired meta-scheduling of parallel applications on global grids, Journal of Parallel and Distributed Computing 73 (4) (2013) 450–464.

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

[37] B. Alrayes, Ö. Kafalı, K. Stathis, Concurrent bilateral negotiation for open e-markets: the CONAN strategy, Knowledge and Information Systems 56 (2) (2017) 463–501.

[38] M. Schillo, C. Kray, K. Fischer, The eager bidder problem: a fundamental problem of dai and selected solutions, in: Proc. of 1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2002, pp. 599–606.

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

[40] Lightbend, Inc, Akka toolkit, https://akka.io, visited 2019-12-17.

[41] J. Bennett, S. Lanning, et al., The netflix prize, in: Proceedings of KDD cup and workshop, Vol. 2007, 2007, p. 35.

[42] Météo France, Données synop essentielles omm, https://donneespubliques.meteofrance.fr/?fond=produit&id_produit=90&id_rubrique=32 (2019).