Fair multi-agent task allocation for large datasets analysis

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

Under consideration for publication in Knowledge and Information Systems

Fair Multi-Agent Task Allocation for Large Data Sets Analysis[1]

Quentin Baert[1]

Anne-Cécile Caron[1]

Maxime Morge[1]

Jean-Christophe Routier[1]

[1]Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

Abstract. MapReduce is a design pattern for processing large datasets distributed on a cluster. Its performances are linked to the data structure and the runtime environment. Indeed, data skew can yield an unfair task allocation, but even when the initial allocation produced by the partition function is well balanced, an unfair allocation can occur during the reduce phase due to the heterogeneous performance of nodes. For these reasons, we propose an adaptive multi-agent system. In our approach, the reducer agents interact during the job and the task re-allocation is based on negotiation in order to decrease the workload of the most loaded reducer and so the runtime. In this paper, we propose and evaluate two negotiation strategies. Finally, we experiment our multi-agent system with real-world datasets over heterogeneous runtime environment.

Keywords: Multi-agent system, Negotiation, Big Data, MapReduce

1. Introduction

Data Science aims at processing large volumes of data to extract knowledge or insights. The volume and velocity of the available data to analyze requires

[1] The final publication is available at Springer via http://dx.doi.org/10.1007/s10115-017-1087-4
Received 16 October 2016
Revised 17 June 2017
Accepted 10 July 2017

parallelizing the processing as it can be done with the MapReduce design pattern (Dean and Ghemawat, 2008). The latter takes its name from the functions on which it is based: the *map* function which filters the data and the *reduce* function which aggregates them. The most popular framework for MapReduce is Hadoop but many other implementations exist, such as the cluster computing framework Spark (Zaharia et al., 2016), or the distributed NoSQL database Riak built from Amazon Dynamo (DeCandia et al., 2007).

The developer of a distributed application based on the MapReduce design pattern must understand the implementation which he uses (e.g. Hadoop), the data to process and the runtime environment in order to better configure the job beforehand. The choice of the implementation and its parametrization can be challenged by a variation in the data or in the runtime environment. In (Kwon et al., 2012*a*), the authors identify four common data skew: two in the mapping phase and two during the reducing phase. The partitioning skew is the one we want to address. According to this data skew, the workload of the reducers is unbalanced and so the reducing phase is penalized by the most loaded reducer. In a similar way, the usage of an heterogeneous cluster environment can lead to a workload unevenly distributed between the reducers.

We show in this paper that Multi-Agent Systems (MAS) are suitable for the distributed implementation of the MapReduce design pattern. In particular, the adaptability of MAS makes it possible to tackle the partitioning data skew and a runtime environment with heterogeneous performances. Our MAS includes two kinds of agents : (i) the mapper agents filter the data; (ii) the reducer agents aggregate the data. In order to balance the workload between reducers, the tasks are dynamically and continuously re-allocated during the reducing phase. The task negotiation fills the gap between the most loaded reducer and the least loaded ones. This balancing of the workload allows to speed up the reducing phase and so the data processing. We prove that the negotiation process terminates and improves the fairness which measures if the processing is performed at the expense of the worst-off agent. During the data processing, each reducer must determine which tasks are locally performed and which ones can be delegated through negotiation. For this purpose, we propose and evaluate here two strategies. Finally, we experiment our multi-agent system with real-world datasets over heterogeneous runtime environment and our observations confirm the added-value of negotiation.[2]

This paper is structured as follows. Section 2 introduces the MapReduce design pattern and the Contract Net Protocol in the background of our work. Section 3 overviews relevant related works. Section 4 describes the core of our proposal. Then, we present in Section 5 our empirical results. Finally, Section 6 concludes with some directions for future work.

[2] This paper is an extension of (Baert et al., 2016). We provide here a deeper description of the framework including some illustrative examples. Finally, we present/evaluate some additional strategies and experiments measuring the speedup of our MAS in a distributed environment.

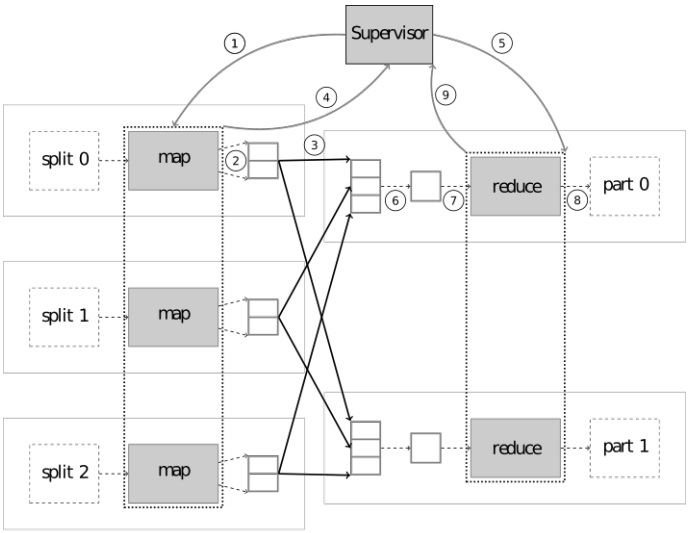


Fig. 1. MapReduce dataflow.

2. Background

2.1. MapReduce

The MapReduce programming model aims at processing large data sets (Dean and Ghemawat, 2008). In this design pattern, while the map function filters data, the reduce function aggregates them.

The distributed implementation Hadoop allows developpers without any experience with parallel and distributed systems to easily use the resources of a large distributed system. MapReduce jobs are divided into a set of map tasks and reduce tasks that are distributed on a cluster of computers. The MapReduce programming model is based on two user-provided functions with the following types:

$$\begin{aligned} map&: \quad (K1, V1) \rightarrow list[(K2, V2)] \\ reduce&: \quad (K2, list[V2]) \rightarrow list[(K3, V3)] \end{aligned}$$

Fig. 1 illustrates the MapReduce dataflow:

1. The supervisor splits the input data among mappers.
2. The mappers apply the map function on the input data and create intermediate key-value pairs ($key : K2$, $value : V2$).
3. A partition function is applied on mappers output in order to split them into subsets, one subset per reducer such that all couples with the same key are grouped and sent to the same reducer. The partition function can be customized in order to specify which keys need to be processed together by the same reducer.
4. Once a mapper has processed its input, it informs the supervisor.
5. When all the mappers have processed their input, the supervisor informs the reducers to start the reduce process.

6. The reducers aggregate the intermediate key-value pairs to build $(K2, list[V2])$ pairs.
7. The reducers take the $(K2, list[V2])$ pairs and run the reduce function once per key grouping.
8. The final key-value pairs $(K3, V3)$ are written into a file in a distributed file system.
9. Finally, the reducers inform the supervisor of the final result location. When all the reducers have finished their tasks, the job is terminated.

Whether a default function or a special one is used, the partitioning is *a priori* fixed. For example, to determine which reducer will process which `key`, the default Hadoop partition performs a modulo of the number of reducers:

`key.hashCode() % numberOfReducers`.

This function, as any *a priori* fixed function without previous knowledge about the data, does not depend on the data, and so the workload is not necessarily uniformly distributed among the reducers.

We have experimented our MapReduce implementation with a weather dataset. This dataset contains more than 3 millions of records (station id, timestamp, temperature, rainfall, ...) from 62 stations taken during the last 20 years. We have written a simple job which counts the number of records per half degree of temperature:

1. the *map* function reads a record $(K1)$ and returns a couple $(K2, V2)$=`(Temp,1)` where `Temp` is the temperature rounded to its nearest half degree for the corresponding station and timestamp.
2. the *reduce* function sums for a given key `Temp` all the counters (always 1) provided by *map* in `s` to produce $(K3, V3)$=`(Temp,s)`.

Fig. 2 shows the outcome of the job on the left and the reducers workload on the right. The reduce task allocation is the result of the default Hadoop partition function. This task allocation is not only the result of a meaningful request based on real-world data but also symptomatic of the use of a predefined partitionning function. Indeed, the job could have been finished earlier if the task allocation would have been balanced. Another partition function could lead to a better key partitionning but, without prior data knowledge, it not possible to anticipate which partition function is suitable.

Even if the partition function produces a balanced initial task allocation, an unfair allocation can occur during the reduce phase due to the heterogeneous performance of nodes. It is worth noticing that the computation time is determined by the most loaded or the slowest process.

Our purpose here is to fix this problem due to data skew or performances heterogeneity by re-allocating the tasks dynamically between reducers during the process. This proposal does not depend on the initial partition function, neither on any data knowledge. Our MapReduce framework is based on a MAS, where reducers are agents negotiating tasks during the job. The next section presents the Contract Net Protocol which our proposal is build on.

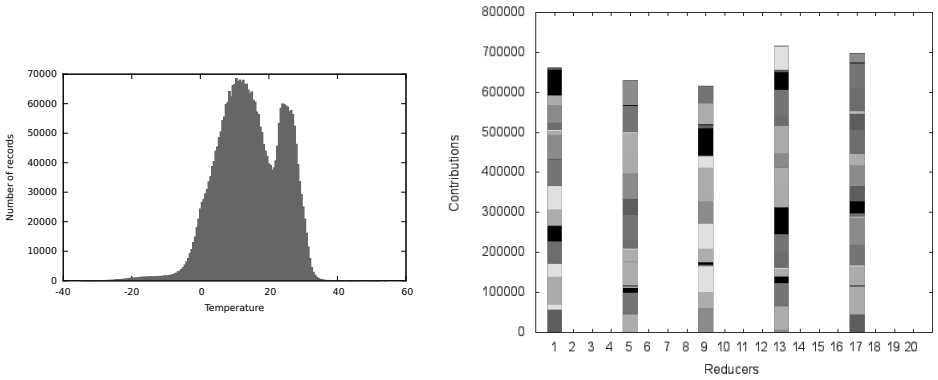


Fig. 2. The number of records per half degree of temperature (at left) and the key partitionning with the default Hadoop partition function (at right).

2.2. Contract-Net Protocol

Introduced by Smith (1980), the Contract Net Protocol (CNP) is the first interaction protocol proposed for task allocation. According to this auction protocol, agents seek to establish some contracts. The CNP is a multiparty protocol where each agent can play the two following roles: the auctioneer (also called initiator) proposes to delegate some tasks while the bidders (also called participants) reply to supply these tasks. In this protocol, the initiator starts by submitting a call for proposal (cfp) to the bidders. Each bidder evaluates this cfp in order to make a proposal. The initiator gathers all the received proposals and allocates the tasks to the bidders which made the best proposals. Afterwards, the selected bidders inform the initiator of the successful processing (or failure) of the tasks. Our multi-agent negotiation protocol is based on the CNP (cf. Section 4.2).

3. Related works

In this paper, we focus on multi-agent negotiation in order to reduce the computation time of the reducing phase (i.e. the workload of the most loaded reducer). Firstly, we review the existing approaches for task partitioning in the reducing phase. Secondly, we highlight the benefit of adaptive MAS, in particular multi-agent negotiation.

Table 1 compares some existing works which have studied the reducing phase optimization.

As stated in Section 2, the key partitioning of Hadoop (Dean and Ghemawat, 2008) is fixed and static. By contrast, Lama and Zhou (2012) and Wolf et al. (2012) predict the performance with job profiling by collecting data from previous runs. In our proposal, we do not want to preprocess data, e.g. with a machine learning phase using a sample dataset. Indeed, we assume no *a priori* knowledge of the input data since such a computational overhead can be high, in particular for large datasets.

SAMR (Slagter et al., 2013) is a scheduling algorithm which uses the history of the previous runs to identify the slow nodes. In this way, this centralized scheduler assigns tasks according to the previous performances of the node. Indeed, we

	Hadoop	SAMR	SkewTune	FP-Hadoop	Our contrib.
No *a priori* knowledge	✓	✗	✓	✓	✓
Dynamic	✗	✗	✓	✓	✓
Tackle data skew	✗	✓	✓	✓	✓
Decentralized	✗	✗	✗	✗	✓
Weak parameterization	✗	✓	✓	✗	✓

Table 1. Features of partitioning data skew tackling contributions.

prefer a dynamic approach where adaptive reducers interact during the job. Our MAS is *per se* adaptive with respect to the computation since the negotiations are performed during the data processing.
Kwon et al. (2012*b*) identify four skews in the MapReduce applications: two of them during the mapping phase (the expensive record skew and the heterogeneous map), and two others during the reducing phase (the partitioning skew and the expensive key group). Then, the authors study unbalanced situations between mappers or between reducers. SkewTune (Kwon et al., 2012*b*) mitigates the partitioning skew due to an uneven distribution of data. When a node is free since it has terminated its task, SkewTune identifies the slowest reducer and reassigns the unprocessed input data. Our approach is similar to SkewTune and we only address here the partitioning skew. However, the partitioning is in our approach the outcome of a collective choice of reducers. Moreover, we think that multi-agent negotiation is suitable to deal with partitioning a set of values associated with the same key (i.e. the expensive key group skew). We plan to address this data skew in future works.
FP-Hadoop (Liroz-Gistau et al., 2016) handles these two data skews with a centralized task split process. They introduce a new intermediate phase which parallelizes the reducing phase for one key. This intermediate phase is managed by the supervisor with a centralized data structure and its parametrization needs a priori knowledge over the data. Our goal is to propose a fully decentralized process where all the decisions are local. We believe that such a decentralized process is more adaptive since it requires less user configuration and handles heterogeneous environments without human supervision.
The mapping phase optimization as studied in (Wolf et al., 2012) is not within the scope of this paper since it is complementary to our approach and it could be implemented by a MAS.
As Table 1 sums up, our contribution is dynamic, decentralized, with neither prior knowledge over the data nor historical data and it does not require data-dependent parameters.

In our work, the dynamic allocation of tasks is based on a negotiation between reducers. Social choice theory provides methods for designing and analyzing collective decision by combining individual preferences or welfares. Computational social choice is often considered as an optimization problem solved by a centralized approach (e.g. an auction) where agents report their preferences to the central and omniscient auctioneer that determines the allocation consequently (Brandt et al., 2016). Indeed, such an approach makes important assumptions that correspond to severe drawbacks : (i) it may be too expensive to gather all information in a single place; (ii) if data evolve during the solving process, it must restart in order to take the new data into account; (iii) it assumes that agents are fully connected without restriction and that they can communicate with all others. We solve here drawbacks (i) and (ii) using multiple distributed concurrent auctions with adaptive agents which only take decisions

based on local information. As we consider fully connected agents, drawback (iii) remains one of our concerns. However, we can easily adapt agents acquaintances network to build subgroups which negotiate independently one from the other. Typically in a distributed system, the communication cost depends on the topology of the network, i.e. physical constraints. A solution would be to adapt the subgroups to the physical network and thus provide suitable negotiations.

Generally speaking, we adopt here a behavioural approach for the distributed problem solving. The approach includes stigmergy (Wagner et al., 1999), distributed constraint solving problem (Clair et al., 2008) or negotiation (Nongaillard and Mathieu, 2011). Unlike (Nongaillard and Mathieu, 2011), our resolution is not about finding a data allocation once for all, but we iterate task negotiations during the reducing phase using a local estimation of the current workload. Contrary to (Clair et al., 2008), the problem which is addressed has no scheduling constraints and the agents have no limited skills or capacity. As stated by Rihawi et al. (2015), the large scale distribution of situated multi-agent system (e.g. ant colony, boids, etc.) is a difficult problem. To the best of our knowledge, the only MAS which implements the MapReduce design pattern is based on mobile agents to ensure code and data replication in order to guarantee fault tolerance (Essa et al., 2014). However, this work does not implement self-organization techniques (Serugendo et al., 2005) to adapt the system to the data or to the computing environment. For this purpose, we adopt multi-agent negotiation techniques.

4. Proposal

We aim at decreasing the workload of the most loaded reducer in order to terminate the reducing phase earlier. For this purpose, we consider dynamic task re-allocation with multi-agent negotiations which do not require a centralized orchestration.

In this section, we present our core proposal. First, we overview the proposal. Second, we show a negotiation example. Third, we present our reducer agent architecture. Fourth, we introduce the different interaction protocols in which reducer agents are involved. Fifth, we detail their behaviours. Finally, we prove some formal properties of our framework.

4.1. Overview

Our contribution aims at providing a balanced reducing task partitioning. For this purpose, we propose a task re-allocation based on local decisions where each reducer is embodied by an agent. Each of them is associated with the bundle of tasks it must achieve. We assume that each task has a cost, i.e. an intrinsic characteristic. Therefore, all the agents, having the same capabilities, estimate their own contributions to the global resolution as the costs of their bundles.

Definition 1 (Allocation/Contribution). Given a set $\mathcal{T}$ of m tasks $\tau_1, \ldots, \tau_m$ with the associated costs $c_{\tau_1}, \ldots, c_{\tau_m}$ and a population $\Omega = \{1, \ldots, n\}$ of n reducer agents, a task *allocation* $\mathcal{A}$ is represented by an ordered list of pairwise

disjoint task bundles $\mathcal{T}_i \subset \mathcal{T}$, such that $\biguplus \mathcal{T}_i = \mathcal{T}$, describing the subset of tasks owned by each agent i:

$$\mathcal{A} = [\mathcal{T}_1, \ldots, \mathcal{T}_n] \text{ with } 1, \ldots, n \in \Omega \tag{1}$$

The *contribution* of the agent i at time t within the allocation $\mathcal{A}$ is defined such that:

$$c_i^{\mathcal{A}}(t) = \sum_{\tau \in \mathcal{T}_i} c_\tau + w_i(t) \tag{2}$$

where $w_i(t)$ is the estimated cost for the current task performed by the agent i. Before the reducing phase, $w_i(0) = 0$.

The mapping phase does not differ from the classical MapReduce model. Mappers deliver intermediate key-values pairs to the reducers. However for each key-values, the mappers add information on the cost of a task for these (partial) values. Since we do not assume any a priori knowledge over the data, the default partitioning is used by all the mappers to achieve the initial allocation to the reducers.

Reducers receive their pairs $(K2, list[V2])$ and start their reduce work. Simultaneously, the negotiation phase begins in order to decrease the contribution of the most loaded reducer, such that the reducing phase finishes earlier. We assumes here that agents are fully connected without restriction and that they can communicate with all others. The reducer agents communicate with each other to negotiate task delegation. Actually, they request their peers through cfp (call-for-proposal) in order to alleviate their contributions. A cfp includes the cost of the submitted task and the auctioneers's contribution.

A reducer bids to take the responsibility of the task in order to decrease the worst contribution. A bidder makes a proposal iff, after the task transfer, the worst resulting contribution is smaller than the worst initial one. Formally, its decision is based on the following local criteria:

Definition 2 (Acceptability criteria). Let $\mathcal{A}$ be an allocation of tasks at time t between n agents Ω. The bidding agent j will make a proposal for the transfer of the task $\tau \in \mathcal{T}_i$ suggested by the auctioneer i iff:

$$c_j^{\mathcal{A}}(t) + c_\tau < c_i^{\mathcal{A}}(t) \tag{3}$$

In other words, a participant agrees to be involved as bidder in a negotiation iff, in case of successful negotiation, its resulting contribution would be strictly smaller than the initial auctioneer contribution. Then, the greatest contribution after the task transfer is smaller than the greatest one before it. It is possible that, after the delegation, the contribution of the bidder becomes greater than the initiator one. This is not an issue since we consider the agents cooperate to solve the problem. Even if one of them works more, they aim at decreasing the contribution of the most loaded agent in order to finish the job earlier. As we will demonstrate latter, the repeated concurrent negotiations lead to decrease the highest contribution.

Reciprocally, the auctioneer can receive several bids replying to its cfp. A bid includes the contribution of the potential supplier. The auctioneer selects the winner with the smallest contribution. Formally,

Definition 3 (Selection criteria). Let $\mathcal{A}$ be an allocation of m tasks $\mathcal{T}$ be-

tween n agents in Ω at time t. If the auctioneer i has proposed to delegate the task τ and it has received some bids from the agents $\Omega' \subset \Omega$, it selects:

$$\text{argmin}(\{c_j^{\mathcal{A}}(t) \mid j \in \Omega'\}) \tag{4}$$

In this way, the task transfer allows to load the least loaded reducer in order to balance the workload as much as possible. It is worth noticing that evaluating the decision criteria for the task transfer only requires local information.

The reducers send cfp as long as their previous cfp has not been denied by all their peers. The protocol ensures that when negotiations stop, there is no task transfer that could lead to a decrease of the maximum contribution. As we will see in Section 4.5, a reducer resumes sending cfp when it acquires knowledge that some of its peers may accept it.

4.2. Example

In order to illustrate the task delegation through negotiation, we consider here a particular auction within a single MapReduce job.

We suppose that the mapping phase has been performed and the reducing tasks are initially allocated to a set of four reducers, $\Omega = \{1, 2, 3, 4\}$. Let us focus on the task allocation at time t ($\mathcal{A} = [\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4]$) such that the individual contributions are: $c_1^{\mathcal{A}}(t) = 10$, $c_2^{\mathcal{A}}(t) = 8$, $c_3^{\mathcal{A}}(t) = 3$ and $c_4^{\mathcal{A}}(t) = 5$ (cf. Fig. 3(a)). In order to decrease their contribution, any agent can initiate an auction. In our example, we focus on the auction initiated by the reducer #1. It sends a cfp about the task $\tau \in \mathcal{T}_1$ with $c_\tau = 3$ (cf. Fig. 3(b)). This cfp includes the reducer #1 contribution $c_1^{\mathcal{A}}(t)$ and the task cost c_τ. Since the reducer #1 has the maximum contribution, this auction may be successful.
In order to reply, the other reducers must decide if they want to take care of the task τ. Using the acceptability criteria (cf. Def. 2) each of these reducer chooses to make a proposal for τ or to decline this task delegation. The reducer #2 does not want to take care of τ, otherwise, its resulting contributions $c_2^{\mathcal{A}}(t) + c_\tau$ would be higher than $c_1^{\mathcal{A}}(t)$. Meanwhile, the reducers #3 and #4 make a proposal for τ by sending their contributions to reducer #1 (cf. Fig. 3(c)).
The reducer #1 must now select the bidder with the lowest contribution. Using the selection criteria (cf. Def. 3), the reducer #1 accepts the proposal from the reducer #3 and rejects the one from the reducer #4 (cf. Fig. 3(d)).

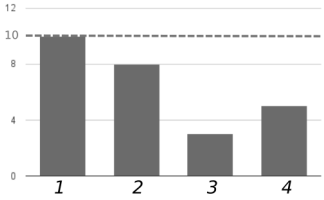
After this negotiation (at time $t + 1$), we observe that:

- the task allocation is $\mathcal{A}' = [\mathcal{T}_1 \setminus \{\tau\}, \mathcal{T}_2, \mathcal{T}_3 \cup \{\tau\}, \mathcal{T}_4]$;
- the contributions are $c_1^{\mathcal{A}'}(t+1) = 7$, $c_2^{\mathcal{A}'}(t+1) = 8$, $c_3^{\mathcal{A}'}(t+1) = 6$ and $c_4^{\mathcal{A}'}(t+1) = 5$.

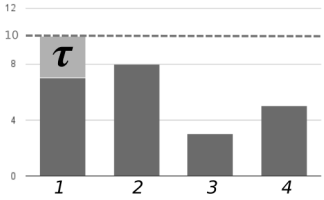
Therefore, the reducer #2 has now the maximum contribution. However, we can observe that the contribution of the most loaded reducer has decreased since we have $c_2^{\mathcal{A}'}(t+1) < c_1^{\mathcal{A}}(t)$. The negotiation leads to a more efficient task allocation when the workload is fairly allocated (cf. Fig. 3(e)).

4.3. Reducer Agent Architecture

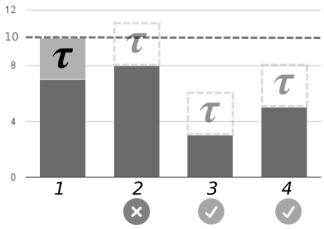
We consider here the asynchronous message-passing model of actor (Clinger, 1981) for concurrent programming. Inspired by Hewitt (1977), we consider that



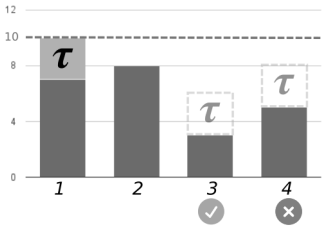
(a) Initial task allocation. The reducer #1 is the most loaded reducer.



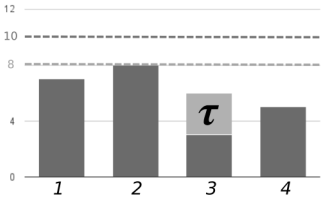
(b) The reducer #1 auctions the task τ.



(c) Each bidder applies the acceptability criteria.



(d) The reducer #1 applies the selection criteria.



(e) Task allocation after the negotiation.

Fig. 3. Step by step negotiation process: how the reducer #1 delegates the task τ.

an agent: (i) has a unique *address*; (ii) is triggered by messages delivered in its mailbox; and (iii) can create other agents.

In order to decrease the complexity related to the design of the reducer agent, we adopt here a recursive agent architecture similar to (Morge et al., 2009)). This modular approach allows: (i) the separation of concerns; (ii) the concurrency of the negotiation phase and the reducing one; (iii) intelligible behaviours. For this purpose, the reducer agent creates three sub-agents which run concurrently (cf. Fig. 4):

1. a worker agent which locally computes several tasks;

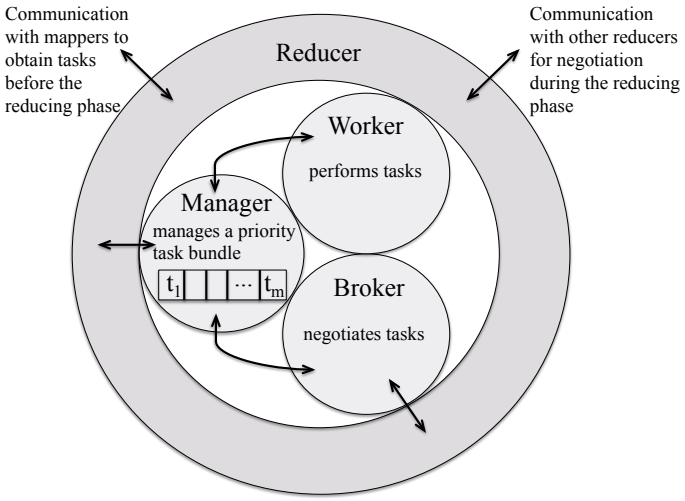


Fig. 4. The reducer agent creates three agents: the manager handling a task bundle; the broker negotiating tasks; and the worker locally performing tasks.

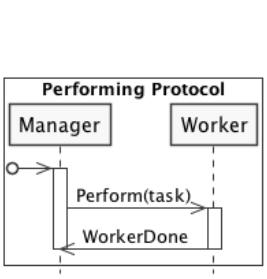
2. a broker agent which negotiates tasks either as a auctioneer or as a broker;
3. a manager agent which orchestrates the negotiations handled by the broker with the tasks performed by the worker. The manager is responsible of the task bundle sorted by the cost of the task. The strategy used to manage the task bundle will be discussed in Section 4.6.

Contrary to the worker agent, the two other ones can both communicate with other agents via their reducer. While the manager agent receives the mapper output, the broker negotiates with other brokers. Actually, the reducer agent plays the role of proxy to forward messages from/toward other agents.

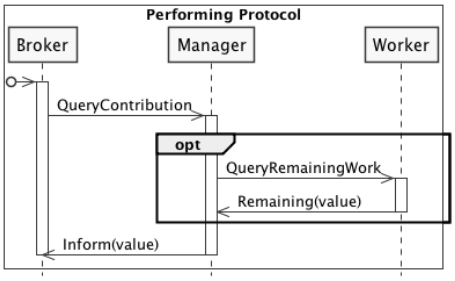
4.4. Protocols

We present here the protocols which regulate the interaction between the sub-agents of the same reducer and, as stated in Section 2, we apply the Contract Net Protocol (Smith, 1980) in order to delegate reducing tasks.

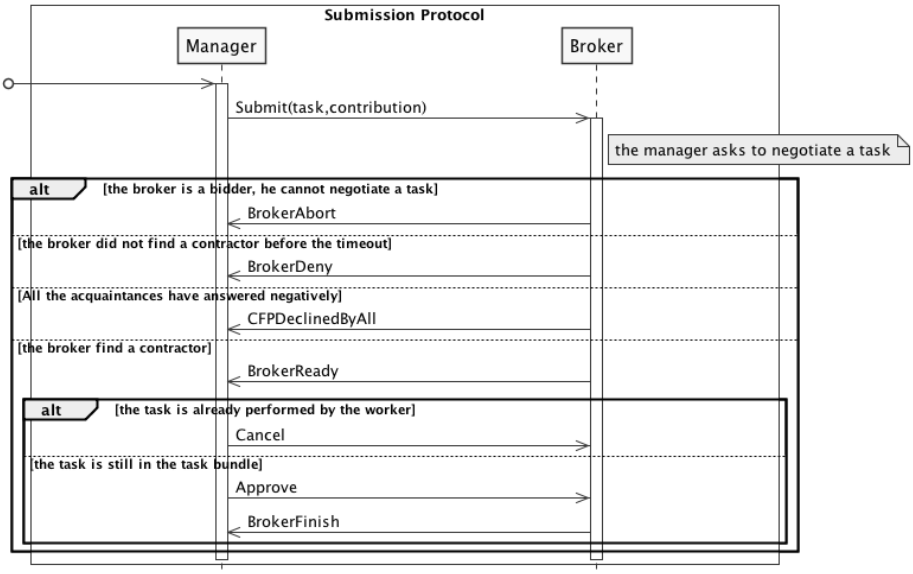
The protocols which regulate the interactions between the sub-agents are depicted in AUML (Odell et al., 2000) within Fig. 5. Within the same reducer, the manager interacts with the worker in order to locally perform some tasks (cf. Fig. 5(a)). The manager assigns a task to the worker through a `Perform` message. When the task is performed, the worker replies with `WorkerDone` and then the manager can send a new task.
The manager interacts with the broker in different ways depending on the role of this latter in the negotiation:



(a) Performing task protocol



(b) Querying contribution protocol



(c) Submiting task protocol

Fig. 5. Protocols regulating interactions between the manager, the worker and the broker of the same reducer agent.

– Either the broker acts as a bidder (cf. Fig. 5(b)), then it needs to know the local contribution in order to reply to a `Cfp`. For this purpose, the broker sends a `QueryContribution` to the manager which replies with `Inform`.

– Or the broker can act as an auctioneer if it is not already involved in another negotiation (cf. Fig. 5(c)). In order to delegate a task, the manager sends a `Submit`. If the broker does not find any potential supplier before the timeout, it replies to the manager with a `BrokerDeny`. If all the agents decline the task delegation, the broker replies to the manager with a `CFPDeclinedByAll`. The broker replies with a `BrokerReady` message when it has found a contractor. Meanwhile the task can have been concurrently performed by the worker. For this reason the manager can `Cancel` the task delegation. Otherwise, the man-

ager sends an `Approve` and the successful delegation ends with a `BrokerFinish` message.

It is worth noticing that these protocols which regulated the interaction between the sub-agents allows to concurrently realize the task delegation through the broker and the task performance through the worker. Even if a broker can be only involved in a single negotiation at a time either as a bidder or as a auctioneer, several auctions involving distinct reducers can simultaneously occur.

As depicted in Fig. 6, an auction is initiated by a broker with a call-for-proposal (`Cfp`) which contains the cost of delegated task and its own contribution. The initiator contribution is broadcast to all the participants. Each participant forward the contribution to its manager with a `InformContribution` message. Depending on its own acceptability criteria (cf. Definition 2) each of the m participants can either decline (`Decline`) or accept the cfp. In the latter case, if it is not already committed in another cfp, the participant sends a `Propose` containing its contribution. Only the proposal with the smallest contribution is selected as the auction winner (cf. Definition 3). The others are notified by a `Reject` while the winner receives an `Accept` with the delegated task and must then definitely acknowledge the delegation with a `Confirm`. It is worth noticing that, since reducers can be distributed within a cluster of PCs, messages are delivered at most once. Since a message can be lost, the auction protocol includes business-level acknowledgements.

4.5. Behaviours

We sketch out the behaviours of the sub-agents which are in conformance with the previous protocols.

Manager. This agent handles the task bundle and coordinates the activities of the worker and the broker. The manager provides some tasks to the worker and bootstraps the broker to initiate auctions. At first, the task bundle is fulfilled by the mappers and will be eventually completed by the broker winning auctions. In order to empty the task bundle, the manager gives priority to the worker. As soon as the worker is free, the manager gives a new task to it. Actually, a task is delegated only if the worker is busy and the manager ensures that the broker is involved in at most one auction. The strategy for the the task bundle management must determine which tasks in the bundle are performed by the worker and which ones can be negotiated. We will discuss these decisions in Section 4.6. Additionally, the manager interacts with the supervisor to detect the termination of the reducing phase. The manager is idle when the worker and the broker are both free and the task bundle is empty. The manager is reactivated when it receives a `Request` from its broker which has won an auction. In this case, the manager must take care of the task delegated by another reducer.

Worker. This agent, which is initially free, becomes busy as soon as it receives a `Perform`. When the task has been performed, the worker informs the manager and it becomes free. During its performance, a worker can be interrupted in order inform the manager about the estimated remaining cost of the current task.

Broker. The broker can act as a bidder or as an auctioneer.
Broker as a bidder. When the broker receives a `Cfp` from another broker (i.e.

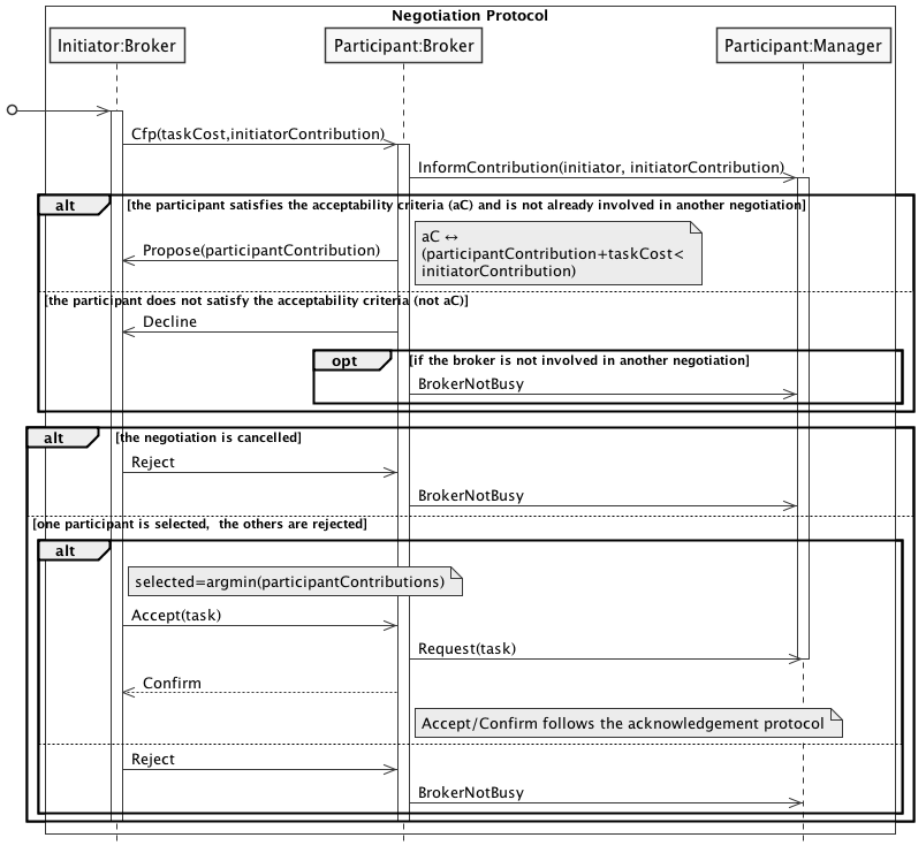


Fig. 6. Negotiation protocol.

another reducer), it queries the local contribution to the manager in order to participate in the auction. If the acceptability criteria (cf. Def. 2) is not fullfilled, then the broker declines the auction. Otherwise, either it does not reply since it is already involved in another auction, or a proposal is sent. When the bidding phase is closed: (i) either the bidder wins the auction and it requests the task to its manager and confirms the task delegation to the auctioneer; (ii) or the bidder does not win the auction (the deadline is reached or `Reject` is received) and the broker informs its manager that it is free.

Broker as an auctioneer. When the manager bootstraps the broker (`Submit`), the latter initiates an auction (`Cfp`). Each reply is recorded, whether it is a `Propose` or a `Decline`. When all of them are received or the deadline is reached, the best proposal is selected (cf. Def. 3). We remark that the negotiation is cancelled and the broker sends an alert (`CFPDeclinedByAll` or a `BrokerDeny`) to its manager if no proposal is received. Otherwise, the broker must accept the winning proposal and reject the other ones. It notifies (`Ready`) the manager that is has found a supplier. Then, the manager tells if the task is still available (`Approve`) or not (`Cancel`). If the negotiated task is no more available (the task has been given to the worker), the winning bid is rejected. Otherwise, the task

is sent to the winner and a confirmation is expected.

We can note that a reducer can expect to delegate a task only if at least one reducer has a lower contribution. Otherwise, the bootstrapping of an auction is useless since the negotiation will failed. We have refined the behaviour of the manager not to initiate such auctions. When an auctioneer receives only `Decline` messages from its peers, the corresponding manager enter in a pause state. In this state, the broker can be triggered as a bidder but it is no more bootstrapped by the manager to initiate an auction. The manager can leave this state only if:

– either this reducer takes care of a new task and so its contribution increases;
– or this reducer is informed that another contribution has decreased due to task delegation or performance.

In other words, the manager leaves the pause state if the acceptability criteria of its peers may become fulfilled. In order to estimate the acceptability criteria of the other reducers, the manager stores and updates information on the other contributions communicated within the `Cfp` messages. We will prove in Section 4.7 that the pause state will be reached simultaneously by all the agents after a finite number of auctions (cf. Prop. 3) and no task transfer can produce a better task allocation in such a case (cf. Prop. 4). This behaviour allows to coordinate the negotiation process and the data processing with local information in order to avoid useless negotiation, and so decreases the communication overhead. It is worth noticing that the task allocation is dynamic and adaptive since negotiation are repeated. If the current task is performed slower than expected by the worker, then an unbalanced allocation will appear, and so the manager may be reactivated by the decreasing of the other contributions in order to delegate the tasks which are still in its bundle.

4.6. Task bundle management

The manager is responsible for the task bundle management. Its strategy must determine which tasks in the bundle are performed by the worker and which ones can be delegated through negotiation. For this purpose, we introduce here two heuristics which will be experimentally compared in Sec 5.

The naive strategy. Intuitively, the cheaper is the task to delegate, the more likely the corresponding auction may be successful. Following this principle, our first strategy consists of locally performing the expensive tasks as much as possible. According to this naive strategy, the manager gives the more expensive tasks to the worker and the cheapest ones to the broker. However, with this approach, the most expensive task will be the last to be considered for delegation and doomed to stay in the bundle until they are performed.

The k-eligible strategy. Our second strategy aims at decreasing the communication overhead. The cheapest task are given to the worker since the cheaper a task is, the higher the communication overhead is. We remark that, in order to decrease the maximum contribution of reducers, the tasks to delegate are not necessarily the cheapest ones. Actually, more expensive tasks can be delegated. Moreover, the negotiation process is iterative and the contributions are communicated within the `Cfp` of the different auctions. In this way, an auctioneer has

Algorithm 1: Selection by the reducer i of the task to delegate.

Data: The maximum number of required potential suppliers k_{max}
Input: the task bundle $\mathcal{T}_i$,
the beliefs of the reducer i about the other contributions $(c_{ij})_{j\in\Omega\setminus\{i\}}$
Output: the task to delegate τ^*

$k \leftarrow k_{max}$;
while $k > 0$ **do**
  $\mathcal{T}_k \leftarrow \{\tau \in \mathcal{T}_i \mid \tau \text{ is k-eligible for the reducer } i\ \}$;
  if $\mathcal{T}_k \neq \emptyset$ **then**
    foreach $\tau \in \mathcal{T}_k$ **do**
      /* Identify the potential reducers which may fulfill the acceptability criteria */
      $\Omega_\tau \leftarrow \{j \in \Omega \setminus \{i\} \mid c_{ij} + c_\tau < c_i\}$;
      /* Choose the worst potential reducer, i.e. having the maximum contribution */
      $\omega_\tau \leftarrow \underset{j\in\Omega_\tau}{\text{argmax}}(c_{ij})$;
    /* Take the task which minimizes the maximum contribution in the worst case */
    $\tau^* \leftarrow \underset{\omega_\tau\in\mathcal{T}_k}{\text{argmin}}(max(c_i - c_\tau, c_{\omega_\tau} + c_\tau))$;
    return τ^*
  else
    $k \leftarrow k - 1$;

some beliefs about the other contributions and it can refine these beliefs during the negotiation process. A reducer can estimate the tasks which can be accepted by its peers.

A task τ is *k-eligible* if the acceptability criteria for τ may be fulfilled by at least k peers.

Definition 4 (k-eligible task). Let $\Omega = \{1, \ldots, n\}$ be a population of n reducers and $\mathcal{A}$ be a task allocation at time t. We denote $(c_{ij}^{\mathcal{A}}(t))_{j\in\Omega\setminus\{i\}}$ the beliefs of the agent i about the other contributions.
The task $\tau \in \mathcal{T}_i$ is *k-eligible* (with $k < n$) for the agent i at time t iff:

$$\exists \Omega' \subseteq \Omega \setminus \{i\}\ (card(\Omega') \geq k \wedge \forall j \in \Omega'\ c_{ij}^{\mathcal{A}}(t) + c_\tau < c_i^{\mathcal{A}}(t)) \tag{5}$$

The more reducers which may fulfill the acceptability criteria for a task, the more likely the corresponding auction will be successful.

Additionally, an auctioneer will select the k-eligible task which minimizes the maximum contribution after the task delegation. In other words, the reducer i selects the task $\tau \in \mathcal{T}_i$ such that τ is a k-eligible task and this task delegation to the reducer ω_τ minimizes the maximum contribution, even if ω_τ is the most loaded potential supplier. This computation is performed by Algo. 1. For this purpose, we introduce the parameter k_{max} which represents the maximum number of expected potential suppliers amongst $\Omega\setminus\{i\}$ ($1 \leq k_{max} < n$). When $k = 1$, all the tasks which may be accepted by at least one agent are (1-)eligible, even if they are expensive. If $k = n-1$, the tasks which may be accepted by all the other reducers are (k-)eligible. The higher k_{max} is, the cheaper the k-eligible tasks are

likely to be. k_{max} avoid to bootstrap auctions about cheap tasks. In this way, we aim at decreasing the communication overhead due to the delegation of cheap task. When ω_τ is initialized (line 7), we consider the most loaded reducer which may make a proposal for τ. In this way, we consider the worst case. The task τ^* minimizes the maximum contribution if the auction is successful (line 8). We can remark $k = 0$ means that no other reducers may make a proposal. In this case, the reducer i switches in the pause state.

In summary, the k-eligible strategy gives the cheapest task to the worker and gives to the broker the k-eligible task which minimizes the maximum contribution if the auction is successful.

4.7. Theoretical results

First of all, we can remark that the negotiation improves the fairness which measures if the processing is performed at the expense of the worst-off agent. The tasks are allocated in a more egalitarian way after a negotiation.

Property 1. The variance of the reducers' contributions strictly decreases after a successful auction.

Proof 1. Let $\Omega = \{1, \ldots, n\}$ be be a population of n reducers. We consider here a successful auction initiated by the reducer 1. For the sake of simplicity, we denote:

–$(c_i)_{i\in\Omega}$, the contributions of the agents before the auction;
–$(c'_i)_{i\in\Omega}$, the contributions of the agents after the auction;
–$\bar{c} = \frac{1}{n}\Sigma_{i=1}^{n} c_i = \frac{1}{n}\Sigma_{i=1}^{n} c'_i$ the mean contribution[10];
–$Var = \Sigma_{i=1}^{n}(c_i - \bar{c})^2$ the variance of the contributions after the auction;
–$Var' = \Sigma_{i=1}^{n}(c'_i - \bar{c})^2$ the variance of the contributions before the auction.

Let $c > 0$ be the cost of the negotiated task and k the reducer which has won the auction. By definition of the negotiation :

1. $c'_1 = c_1 - c$
2. $c'_k = c_k + c$

Due to the acceptability criteria of the reducer k (cf. Def. 2), $c_k + c < c_1$, and so $c_k + c - c_1 < 0$.

$$Var' = \Sigma_{i=2, i\neq k}^{n}(c'_i - \bar{c})^2 + ((c_1 - c) - \bar{c})^2 + ((c_k + c) - \bar{c})^2$$

Then,

$$\begin{aligned} Var' - Var &= ((c_1 - c) - \bar{c})^2 + ((c_k + c) - \bar{c})^2 - [(c_1 - \bar{c})^2 + (c_k - \bar{c})^2] \\ &= ((c_1 - \bar{c}) - c)^2 + ((c_k - \bar{c}) + c)^2 - [(c_1 - \bar{c})^2 + (c_k - \bar{c})^2] \\ &= -2 \times (c_1 - \bar{c}) \times c + 2 \times (c_k - \bar{c}) \times c + 2 \times c^2 \\ &= 2 \times c \times (c_k + c - c_1) \end{aligned}$$

Then, $Var' - Var < 0$.

[10] It is worth noticing that the task delegation is conservative.

It is worth noticing that the whole negotiation process also improves the fairness.

Property 2. The variance of the contributions strictly decreases due to successful iterated and concurrent auctions.

Proof 2. The behaviour of the manager ensures that a reducer cannot be simultaneously involved in several auctions (cf. Section 4.5). Actually, the role of bidder and auctioneer are mutually exclusive. When a broker plays the role of an auctioneer, it does reply to the other auctions as a bidder. Reciprocally, a broker cannot be bootstrapped as an auctioneer when it is involved as a bidder in another auction. The concurrent auctions involve disjoint groups of reducers. It results that each auction is independent. In other words, the outcomes of auctions do not impact one another. According to Prop. 1, each successful auction strictly decreases the variance of contributions. Therefore, the iterated and concurrent execution of such auctions also strictly decreases the variance of contributions.

The whole negotiation process terminates.

Property 3. The iteration of successful negotiations terminates.

Proof 3. Acccording to Prop. 2, the variance is positive and it strictly decreases during the whole negotiation process. Moreover, the number of tasks is finite. Therefore, after a finite number of auctions, the variance of contributions stops to decrease when no more successful auction is possible.

The negotiation process is sound. When it halts, no other task tranfer can alleviate the most loaded reducer.

Property 4. When the whole negotiation process terminates, there exists no task transfer which can decrease the contribution of the most loaded agent.

Proof 4. Let $\Omega = \{1, \ldots, n\}$ be a population of n reducers and $\mathcal{A}$ be a task allocation at time t. Let j be the most loaded reducer and τ be the cheapest task in $\mathcal{T}_j$. We assume that there exists a reducer i which can alleviate j, i.e. $c_i + c_\tau < c_j$. In this case, the acceptability criteria for the reducer i is fulfilled. The corresponding auction would be successful which is a contradiction.

This properties will be experimentally validated in the next section.

5. Experimentations

Our experiments aim at: (i) comparing our two strategies for the task bundle management; (ii) evaluating our proposal with respect to the classical distributed MapReduce programming model.

Our prototype implements the classic and the adaptive distribution of the MapReduce programming model (Dean and Ghemawat, 2008). It is developed with the programming language Scala[12] and the Akka toolkit[13]. The latter, based on the actor model (Hewitt, 1977), helps to fill the gap between the specification of the agent behaviours (cf. Section 4.5) and their implementations and to deploy

[12] http://www.scala-lang.org/
[13] http://akka.io

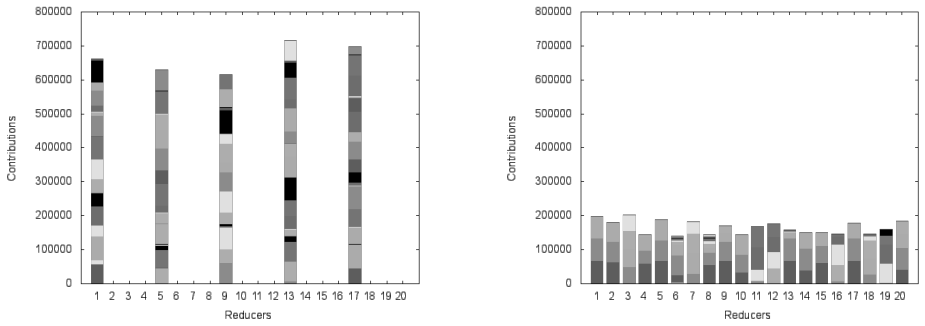


Fig. 7. The contributions of reducers for the classic MapReduce (at left) and for our multi-agent system (at right) with the job `RecByTemp`.

them on a cluster of PCs. In particular, the three sub-agents of a reducer always run on the same node and so message passing between these sub-agents is as fast as method invocation.

5.1. Task bundle management

We make the assumption that the k-eligible strategy decreases the communication overhead with respect to the naive strategy (cf. Section 4.6). In order to empirically validate this hypothesis, we measure the number of auctions which induce a communication cost. Obviously, we expect that the final task allocation reached by the k-eligible strategy is as good as that reached by the naive strategy. For this purpose, we consider the fairness, i.e. the ratio between the running time of the first reducer to end its work and the running time of the last reducer to end its work. This metric indicates if the processing is performed at the expense of the worst-off reducer. If the measure is closed to 1, the allocation is fair. For each set of parameters, we perform 5 runs. Since the standard deviation due to the non-determinism of the scheduler is low, we only exhibit the means of the measured metrics over the different runs.

We analyze here a real-world dataset which contains more than 3 million weather records (station id, timestamp, temperature, rainfall, ...) from 62 stations taken during the last 20 years. We consider here two different meaningful jobs:

1. the first one `RecByTemp` (which stands for *records by temperature*) counts the number of records per half degree of temperature. This job is performed by 10 mappers and 20 reducers.. Some reducing tasks are small, others are large. The task allocation performed by the default Hadoop partitioning is unfair (cf. Fig. 7).
2. the second one `RainByStat` (which stands for *rainfall by station*) counts the accumulated rainfall per station. This job is performed by 10 mappers and 10 reducers. The size of the reducing tasks are homogeneous and the default task partition is almost fair (cf. Fig. 8).

Since they are running on a single multi-core dedicated computer, we can assume that the performances of our reducers are homogeneous in time.

Table 2 presents the empirical results for the job `RecByTemp`. In order to have

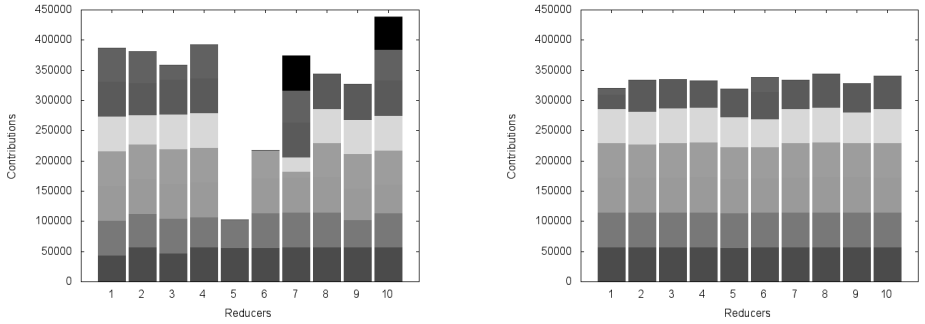


Fig. 8. The contributions of reducers for the classic MapReduce (at left) and for our multi-agent system (at right) with the job `RainByStat`.

	The naive strategy	**The k-eligible strategy**
Total number of auctions	1383	265
Percentage of successful auctions	24 %	22 %
Fairness	0.8	0.85

Table 2. Comparison of the naive strategy and the k-eligible strategy with the job `RecByTemp`.

enough successful auctions, we fix the maximum number of required potential suppliers, i.e. $k_{max} = 4$. In other words, a task is negotiated if at least 20 % of the reducers make a proposal for it. We observe that the k-eligible strategy leads to one fifth of the auctions of the naive strategy and that task allocation is slightly fairer.

Table 3 shows the empirical results for the job `RainByStat`. We fix $k_{max} = 2$ for similar reasons as previously. Since the initial task allocation performed by the default Hadoop partition is almost fair, we observe that much fewer auctions are required than with the job `RecByTemp`. However, the k-eligible strategy still significantly decreases the number of auctions.

In both cases, the percentage of successful auctions is low whatever the strategy is. Indeed, few reducers having a small contribution can act as bidders in the different auctions. Unfortunately, they cannot be involved simultaneously in several auctions. This is why the number of unsuccessful auctions remains high. Nevertheless, we observe that the k-eligible strategy requires significantly fewer auctions to balance the workload. We have empirically validated our hypothesis. Moreover, the adoption of the k-eligible strategy does not penalize the fairness of the task allocation. Small deviations can be explained by the shape of the dataset. That are the reasons why we select the k-eligible strategy in order to compare our adaptive multi-agent system with the classical distribution of the MapReduce programming model.

	The naive strategy	**The k-eligible strategy**
Total number of auctions	87	48
Percentage of successful auctions	16 %	16 %
Fairness	0.89	0.91

Table 3. Comparison of the naive strategy and the k-eligible strategy with the job `RainByStat`.

5.2. Task re-allocation

Our experiments aim at evaluating our proposal with respect to the classical distributed MapReduce programming model. Let us remember that the initial task allocation for our MAS corresponds to the default Hadoop partition.

Yahoo! operates an auction-based platform for selling advertising space next to Yahoo! Search results. Advertisers bid for the right to appear alongside the results of particular search queries. For example, a travel vendor might bid for the right to appear alongside the results of the search query "Las Vegas travel". An advertiser's bid is the price the advertiser is willing to pay whenever a user actually clicks on their ads. We analyze the dataset corresponding to the period from the 15th June, 2002 to the 14th June, 2003, which contains 77 10^6 bids (day, advertiser ID, list of keywords, etc.), i.e. 8 Go[15]. Since a bid is related to a list of keywords, we consider the job `yahooCountByKeyword` which counts the number of bids for each keyword.

We make the assumption that balancing the workload between reducers decreases the running time of the reducing phase. For this purpose, we compare the running time of the reducing phase in the classical distributed MapReduce programming model with our MAS. We run the job `yahooCountByKeyword` with 10 mappers and 10 reducers ($k_{max} = 2$). We present the running times according to the number of PCs used, i.e. Intel (R) Core (TM) i5 3.30GHz PCs with 4 cores and 8GB of RAM. For each set of parameters, we perform 3 runs. Since the standard deviation due to the non-determinism of the scheduler is low, we only show the averages on these runs.

Figure 9 shows the running times of the different phases. Figure 10 highlights the corresponding fairness. We observe that the running time of the mapping phase decreases with the number of PCs since it benefits from the parallelism. Whatever the approach is, the running time of the reducing phase globally decreases. This decline is not perfectly proportional to the number of machines since the reducing phase is penalized by the non-locality of the data: a reducer can process data from a mapper which is on another PC. Moreover, the classic approach is penalized by the data skew. Since it does not adapt itself to the unbalanced partitioning, this approach does not fill the gap between the effort made by the most loaded reducer and those which are less. In the classic approach, fairness remains low as shown in Figure 10: the least loaded reducer works around 50% less than the most loaded one. Conversely, the adaptive approach finish earlier. This time saving is explained by a better exploitation of the available resources, i.e. a fairness closed to 1 which means that the job is evenly distributed between all the reducers. Indeed, the negotiation allows us to dynamically and continuously distribute the tasks to the least loaded reducers.

Additionally, we have adopted the two approaches to perform the job `yahooCountByKeyword` on 2 heterogeneous computers, i.e. an Intel (R) Core (TM) i5 3.3GHz PC with 4 cores and 8GB of RAM and an Intel (R) Core (TM) i7 2.8GHz MacBook Pro with 8 cores and 16GB of RAM. 5 reducers run on each of these computers. The adaptive approach speeds up the the running time of the reducing phase with respect to the classic approach. Left of figure 11 shows the initial task allocation performed by the classic approach. In our approach, this initial allocation is adapted to the heterogeneous performance of nodes (cf

[15] `http://webscope.sandbox.yahoo.com/`

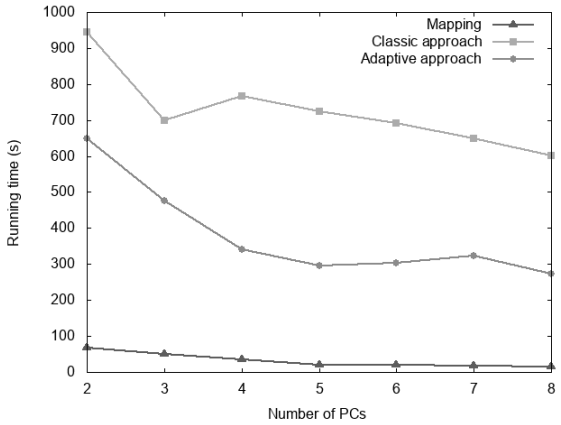


Fig. 9. Running time of the different phases for the job `yahooCountByKeyword`

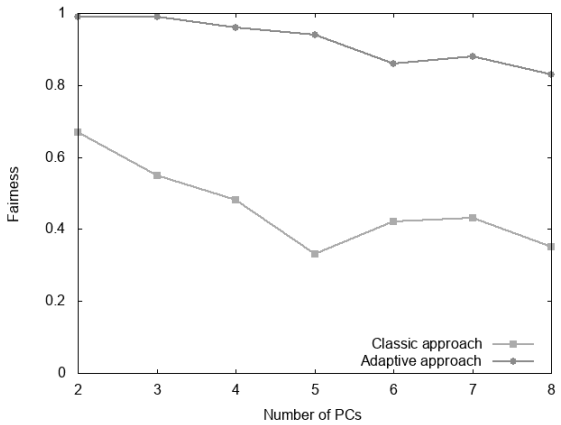


Fig. 10. Fairness between reducers for the job `yahooCountByKeyword`

right of figure 11). The dynamic and continuous task reallocation loads more the first 5 reducers which run on the faster computer.

These experiments show that our MAS benefits from the parallelism better than the classical approach. Due to the partitioning skew or the heterogeneity of the cluster, the negotiation of the workload between reducers decreases the running time of the reducing phase.

6. Discussion

MapReduce applications are complex to optimize because they are based on user-defined operations and the programmer need to understand the implementation of the framework (for instance, Hadoop). In particular, data skew can lead to an uneven workload balancing since the key partitioning is statically fixed. Unbalancing can also occur during the job processing because of the node's per-

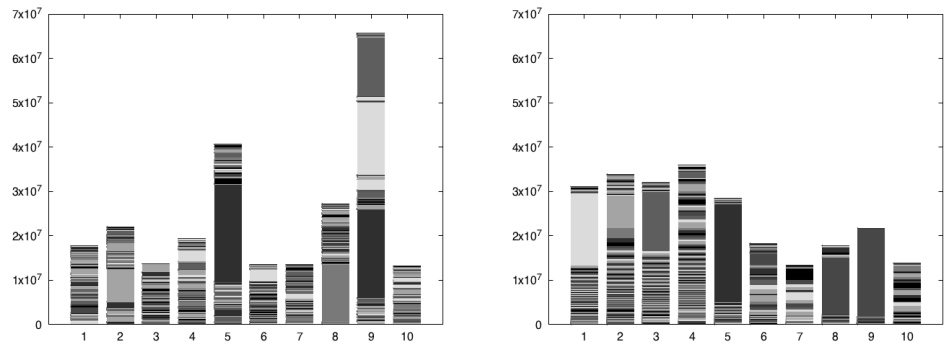


Fig. 11. The contributions of reducers for the classic MapReduce (at left) and for our multi-agent system (at right) with the job `yahooCountByKeyword`. Nodes are heterogeneous: 5 left reducers run faster than 5 right reducers.

formances. Our MAS consists of a distributed model of computation which is inherently adaptive. In this paper, we have defined a distribution of the MapReduce program model where the task allocation is the outcome of multi-agent negotiations during the reducing phase. This dynamic task re-allocation is based on local decisions of reducers embodied by agents, with neither global orchestration, nor data preprocessing, nor data-dependent parameters. More precisely, each reducer is composed of three sub-agents which run concurrently: a worker which locally performs tasks; a broker which negotiates them; and a manager which coordinates the negotiation and the local data processing. In order to balance the workload, these agents negotiate tasks based on their individual contributions in order to decrease the contribution of the worst-off reducers, i.e. the one which delays the reducing phase. We have proved that the negotiation process improves the fairness of the task allocation. Our experiments has confirmed that our proposal decreases the running time of the reducing phase. The workload is dynamically and continuously adapted to be more fairly distributed among reducers, and so decreases the impact due to the partitioning skew or the nodes' heterogeneity. Additionally, we have studied two strategies for the task bundle management. We have selected the k-eligible strategy since our experiments have shown that this strategy requires significantly fewer auctions to balance the workload without significantly penalizing the fairness of the task allocation.

We consider several perspectives for this work.

In the short term, we plan to improve our framework. First, we want to allow a reducer to be a bidder in several concurrent auctions. This extension must preserve the soundness and the termination of the negotiation process but it should also increase the rate of successful auctions. Therefore, the communication overhead will decrease. Second, we want to address the problem of expensive key group. Fig. 12 illustrates that the negotiation process stops when the tasks are too large to be delegated. For this purpose, we aim at considering divisible tasks. If a large task is split in smaller subtasks, the negotiation of these latter will allow us to reach a fairer task allocation.

In the long term, we plan to take into account not only the number of operations to perform but also the data locality in the task cost. In this perspective, the task cost will depend on the reducer which evaluates it.

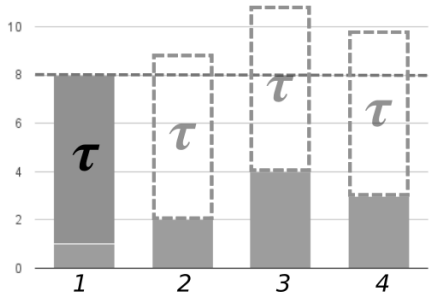


Fig. 12. The reducer # 1 cannot delegate the task τ if this latter is indivisible.

Acknowledgement

This project is supported by the CNRS Challenge Mastodons. The authors would like to thank the scientific committee of the International Conference on Practical Applications on Multi-agent Systems for the invitation in this special issue and the reviewers for their useful suggestions.

References

Baert, Q., Caron, A.-C., Morge, M. and Routier, J.-C. (2016), Fair multi-agent task allocation for large data sets analysis, *in* 'Proc. of 14th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS), Sevilla, Spain.', Vol. 9662 of *LNAI*, Springer, pp. 24–35.

Brandt, F., Conitzer, V., Endriss, U., Lang, J. and Procaccia, A. D. (2016), *Handbook of Computational Social Choice*, Cambridge University Press.

Clair, G., Kaddoum, E., Gleizes, M. P. and Picard, G. (2008), Self-regulation in self-organising multi-agent systems for adaptive and intelligent manufacturing control, *in* 'Proc. of the Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2008, 20-24 October 2008, Venice, Italy', pp. 107–116.

Clinger, W. D. (1981), Foundations of actor semantics, PhD thesis, Massachusetts Institute of Technology.

Dean, J. and Ghemawat, S. (2008), 'Mapreduce: simplified data processing on large clusters', *Commun. ACM* **51**(1), 107–113.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. (2007), Dynamo: Amazon's highly available key-value store, *in* 'Proc. of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)', pp. 205–220.

Essa, Y. M., Attiya, G. and El-Sayed, A. (2014), 'Mobile agent based new framework for improving big data analysis', *International Journal of Advanced Computer Science and Applications* **5**(3), 25–32.

Hewitt, C. (1977), 'Viewing control structures as patterns of passing messages', *Artif. Intell.* **8**(3), 323–364.

Kwon, Y., Balazinska, M., Howe, B. and Rolia, J. (2012*a*), Skewtune: Mitigating skew in mapreduce applications, *in* 'Proc. of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD'12', pp. 25–36.

Kwon, Y., Balazinska, M., Howe, B. and Rolia, J. A. (2012*b*), 'Skewtune in action: Mitigating skew in mapreduce applications', *PVLDB* **5**(12), 1934–1937.

Lama, P. and Zhou, X. (2012), Aroma: Automated resource allocation and configuration of

mapreduce environment in the cloud, *in* 'Proc. of the 9th Internatinal Conference on Autonomic Computing (ICAC'12)', pp. 63–72.

Liroz-Gistau, M., Akbarinia, R., Agrawal, D. and Valduriez, P. (2016), 'FP-Hadoop: Efficient processing of skewed mapreduce jobs', *Information Systems* **60**, 69–84.

Morge, M., Stathis, K. and Vercouter, L. (2009), Arguing over motivations within the V3A-architecture for self-adaptation, *in* 'Proc. of the International Conference on Agents and Artificial Intelligence (ICAART), Porto, Portugal, January 19 - 21', pp. 214–219.

Nongaillard, A. and Mathieu, P. (2011), 'Egalitarian negotiations in agent societies', *Applied Artificial Intelligence* **25**(9), 799–821.

Odell, J., Parunak, H. V. D. and Bauer, B. (2000), Extending UML for agents, *in* G. Wagner, Y. Lesperance and E. Yu, eds, 'Proc. Of the Agent-Oriented Information Systems Workshop at the 17 th National Conference on Artificial Intelligence'.

Rihawi, O., Secq, Y. and Mathieu, P. (2015), Load-balancing for large scale situated agent-based simulations, *in* 'Proc. of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, 1-3 June, 2015, 2014', pp. 90–99.

Serugendo, G. D. M., Gleizes, M.-P. and Karageorgos, A. (2005), 'Self-organization in multi-agent systems', *The Knowledge Engineering Review* **20**(02), 165–189.

Slagter, K., Hsu, C., Chung, Y. and Zhang, D. (2013), 'An improved partitioning mechanism for optimizing massive data analysis using mapreduce', *The Journal of Supercomputing* **66**(1), 539–555.

Smith, R. (1980), 'The contract net protocol: Highlevel communication and control in a distributed problem solver', *IEEE Trans. on Computers, C* **29**, 12.

Wagner, I. A., Lindenbaum, M. and Bruckstein, A. M. (1999), 'Distributed covering by ant-robots using evaporating traces', *IEEE Trans. Robotics and Automation* **15**(5), 918–933.

Wolf, J. L., Balmin, A., Rajan, D., Hildrum, K., Khandekar, R., Parekh, S., Wu, K. and Vernica, R. (2012), 'On the optimization of schedules for mapreduce workloads in the presence of shared scans', *VLDB J.* **21**(5), 589–609.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S. and Stoica, I. (2016), 'Apache spark: a unified engine for big data processing', *Commun. ACM* **59**(11), 56–65.

Author Biographies

Quentin Baert has received a Master degree in Computer Science from University of Lille in 2016. He is currently a Ph.D. candidate. His research activities are related to Computer Science, in particular Artificial Intelligence and Data Science. He aims at proposing decentralized and intelligent computational processes for analysing large volumes of data to extract knowledge or insights.

Anne-Cécile Caron is currently Associate Professor at University of Lille since 1993. She is responsible of the Master of Business Informatics. Her research activities are related to Computer Science, in particular Database Management Systems and Formal Languages.

Maxime Morge is Associate Professor at University of Lille since 2006. He is responsible of the Master curriculum "Business Intelligence Management Skills Training". He was Research Associate at University of Pisa from 2006 until 2009 and he has defended his PhD thesis in 2005 at MINES Saint-Étienne. His research activities are related to Computer Science, in particular Artificial Intelligence, Argumentation and Social Choice Theory.

Jean-Christophe Routier is currently Professor at University of Lille since 2007. He has been Associate Professor at University of Lille since his PhD thesis in 1994. His research activities are related to Computer Science, in particular Artificial Intelligence and Multiagent simulation.

Correspondence and offprint requests to: Maxime Morge, University of Lille, F-59000 Lille, France. Email: Maxime.Morge@univ-lille.fr