Fair Multi-agent Task Allocation for Large Data Sets Analysis

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

Fair Multi-Agent Task Allocation for Large Data Sets Analysis*

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

Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France
quentin.baert@etudiant.univ-lille1.fr,
{anne-cecile.caron,maxime.morge,jean-christophe.routier}@univ-lille1.fr

Abstract. Many companies are using MapReduce applications to process very large amounts of data. Static optimization of such applications is complex because they are based on user-defined operations, called map and reduce, which prevents some algebraic optimization. In order to optimize the task allocation, several systems collect data from previous runs and predict the performance doing job profiling. However they are not effective during the learning phase, or when a new type of job or data set appears. In this paper, we present an adaptive multi-agent system for large data sets analysis with MapReduce. We do not preprocess data and we adopt a dynamic approach, where the reducer agents interact during the job. In order to decrease the workload of the most loaded reducer - and so the execution time - we propose a task re-allocation based on negotiation.

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

1 Introduction

Data Science aims at processing large volumes of data to extract knowledge or insights. Since the technological potential and the societal demand increase, new methods, models, systems and algorithms are developed. The volume and velocity of available data requires new forms of processing to enable their analysis. For this reason the MapReduce design pattern [1] is very successful. The most popular MapReduce framework is Hadoop, but numerous implementations exist, as the cluster computing framework Spark [2], or the distributed NoSQL database Riak built from Amazon Dynamo [3]. In these approaches, the extraction and processing techniques are distributed and operated without sampling.

Data flows can have periodic (daily, weekly or seasonal) and event-triggered peak data loads. These peaks can be challenging to manage. In the existing frameworks, an efficient task distribution (i.e. the key partitioning) requires prior

* This work is part of the PartENS research project supported by the Nord-Pas de Calais region (researcher/citizen research projects).

knowledge of the data distribution. The partitioning is *a priori* fixed and so the workload is not necessarily uniformly distributed. By contrast, multi-agent systems are inherently adaptive and thus particularly suitable when workloads constantly evolve.

In this paper, we propose an adaptive multi-agent system for large data sets analysis based on the MapReduce paradigm. The data processing is distributed among two kinds of agents : i) the mapper agents filter data; ii) the reducer agents aggregate the data. In order to balance the workload between reducers, the tasks are dynamically re-allocated among reducers during the process without sampling. For this purpose, reducers are involved in multiple concurrent auctions. Our agents negotiate tasks based on their individual workload in order to decrease the workload of the worst-off agent, i.e. the one which delays 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. We have experimented our multiagent system over real-world data and our observations confirm the added-value of negotiation.

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

2 Related works

In [1], the authors present the MapReduce programming model and its implementation for processing large data sets. In this model, while the map function plays the role of filtering data, the reduce function aggregates data. It allows programmers 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 calls for two user-provided functions with the following types:

$$map\text{: } (K1, V1) \rightarrow list[(K2, V2)]$$
$$reduce\text{: } (K2, list[V2]) \rightarrow list[(K3, V3)]$$

The partitioner takes the intermediate key-value pairs from the mapper and splits them into subsets, one subset per reducer such that all values associated with the same key $K2$ are grouped into a sequence and passed to a reducer. By default the partitioner performs a modulo operation of the number of reducers (`key.hashCode() % numberOfReducers`) whatever the used partitioner (e.g. `YARN` or `MESOS`). Additionally, the partitioner can be customized in order to specify which keys need to be processed together in a single reducer. In this way, the reducer takes pairs of the form $(K2, list[V2])$ and run the reduce function once per key grouping. Once it is done, the final key-value pairs $(K3, V3)$ are written to a file.

Whether a default function or a special one is used, the partitioning is *a priori* fixed. It means that this function does not depend on the data, and so the

workload is not necessarily uniformly distributed among the reducers. It results that the computation time is determined by the most loaded reducer.

In this paper, we focus on reducing the workload of the most loaded reducer. Several systems have studied the reduce phase optimization, for instance by predicting the performance with job profiling, collecting data from previous runs (see [4] or [5]). We do not want to preprocess data (e.g. with a machine learning phase using a sample dataset) and we prefer a dynamic approach where adaptive reducers interact during the job. In [6], the authors study unbalanced situations between mappers or between reducers. They design a system named SkewTune, which mitigates two types of skew : skew due to an uneven distribution of data and skew due to some subsets of the data taking longer to process than others. When a resource is available because its task is completed, SkewTune identifies the slowest reducer and re-partitions its unprocessed input data. Our approach is similar but, by contrast, we want the partitioning to be a collective choice of workers. Moreover, we intend to deal with partitioning a set of values associated with the same key, and not only partitioning a set of keys. In [7], the authors propose adaptive mappers to decrease the startup overhead. Such optimization is complementary to our approach and could be implemented by a MAS.

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 [8]. 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. Typically in a distributed system, the communication cost depends on the topology of the network, i.e. physical constraints. We consider here multiple distributed concurrent auctions. By contrast, [9] considers MRF in the domain application of UAV where the underlying assumptions are quite different. For instance, the acquaintance network is highly dynamic and the cost of tasks are different from one agent to another.

3 Proposal

We aim at reducing the workload of the most loaded reducer. For this purpose, we consider dynamic task re-allocation with a multi-agent system which does not require a centralized supervision.

In this section, we present our core proposal. First, we overview the proposal. Second, we present our reducer agent architecture. Third, we introduce the different interaction protocols in which reducer agents are involved. Fourth, we detail their behaviour. Finally, we present some formal properties.

3.1 Overview

Our contribution aims at providing a balanced reducer tasks partitioning. In this purpose we propose a task re-allocation based on local decisions where each reducer is embodied by an agent. This agent is characterized by the bundle of tasks it must achieve. We assume that each task has a cost, i.e. an intrinsic characteristic. Therefore, all the agents, with 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 *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:*

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

The contribution *of the agent i at time t within the allocation A is defined such that:*

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

where w_i is the estimated cost of the work-in-progress of agent i. Before starting the reduce phase, $w_i(0) = 0$.

Mapper 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. The default partitioning is then used to achieve the initial distribution 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. 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 proposer'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 A be an allocation of tasks at time t between n agents Ω. The agent j will accept the transfer of the task $\tau \in \mathcal{T}_i$ from i iff:*

$$c_j^A(t) + c_\tau < c_i^A(t)$$

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 initiator contribution. Then, for the two involved agents, the greatest contribution after the transfer is smaller than the greatest one before. It results that through repeated negotiations, the highest contributions decrease then the most loaded agents will finish its tasks earlier.

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

Definition 3 (Selection criteria). *Let A be an allocation of m tasks $\mathcal{T}$ between n agents in Ω at time t. If the agent i has proposed to delegate the task τ and it has received some bids from the agents $\Omega' \subset \Omega$, it selects:*

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

In this way, the task transfer allows to load the least loaded reducer in order to balance the workload. 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 acquaintances. The protocol ensures that when negotiations stop, there is no task transfer that could lead to a decrease of the highest contribution. A reducer resumes sending cfp when it acquires knowledge that some of its acquaintances are liable to accept it.

3.2 Reducer Agent Architecture

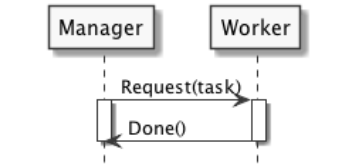
Inspired by [10], we consider that an agent: i) has a unique *id*; ii) is triggered by messages delivered in its mailbox; and iii) can create other agents. The reducer agent creates three agents:

1. a worker agent which locally computes several tasks;
2. a broker agent which negotiates tasks in order to delegate them and potentially adopt additional ones;
3. a manager agent which is responsible of the task bundle to be distributed between the worker and the broker. The task bundle is sorted based on the task costs. In order to increase the likelihood to find a supplier, the manager tries to delegate the task with the lowest cost. By contrast, the task with the highest cost is locally performed by the worker.

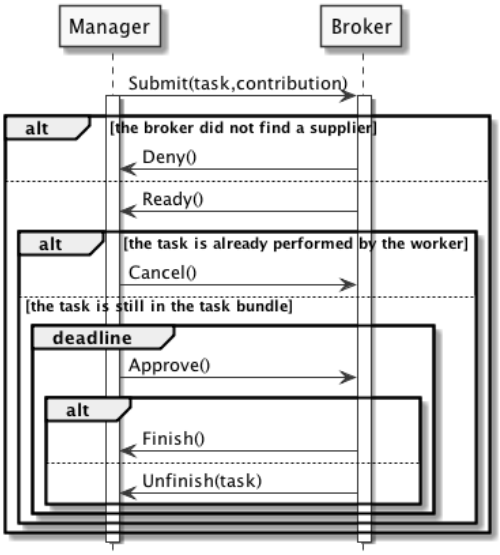
Contrary to the worker agent, the two other ones can both communicate with other agents via their reducer. While the manager agent receives the mapper output, the broker negotiates with other brokers. Actually, the reducer agent plays the role of proxy to forward messages from/toward other agents.

3.3 Protocols

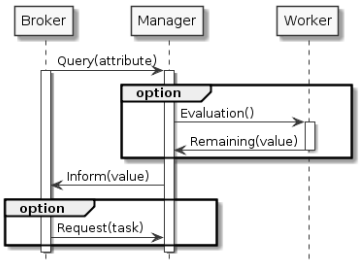
The manager interacts with the worker in order to locally perform some tasks (cf Fig. 1a). The manager assigns a task to the worker through a `Request` message and the worker replies with `Done` when the task is performed. Then, the manager is able to send a new task. In order to know the estimated cost of the work in progress, the manager can also send an `Evaluation` message to the worker, and the worker replies with `Remaining`.



(a) Request protocol



(b) Submit protocol



(c) Query protocol

Fig. 1: Protocols regulating interactions between the manager, the worker and the broker of the same reducer agent

The manager interacts with the broker in different ways depending on its role in the negotiations : a broker can be either the initiator of a negotiation or it can be one of the bidders.

If the broker acts as a bidder (cf Fig. 1c), then it needs to know the local contribution in order to reply to a `Cfp`. For this purpose, the broker sends a `Query` to the manager which replies with an `Inform`. Eventually, the broker can request to the manager a task to perform if it has won the auction. In this case, this task is added to the bundle.

To delegate a task the manager sends a `Submit`, then the broker initiates a negotiation (cf Fig. 1b). If the broker does not find any potential supplier,

it replies to the manager with a `Deny`. Otherwise, the broker replies with a `Ready` message. In the latter case, it is still possible that meanwhile the manager has given the task, which had been submitted, to the worker. For this reason the manager can `Cancel` it. Otherwise, the manager sends an `Approve` which confirms the successful delegation with a `Finish` message. If it is not the case, the manager receives an `Unfinish` message and the task returns to the bundle.

Finally, brokers can negotiate a task delegation through an auction (cf Fig. 2). Such a negotiation is initiated by a broker with a call-for-proposal (`Cfp`) which contains the cost of delegated task and its own contribution. Depending on its own acceptability criteria (cf Def. 2) each of the m participants can either decline (`Decline`) or accept the cfp. In the latter case, the participant sends a `Propose` containing its contribution. Only the proposal with the smallest contribution is selected as the auction winner (cf Def. 3). The others are notified by a `Reject` while the winner receives an `Accept` and must then definitely acknowledge the delegation with a `Confirm`.

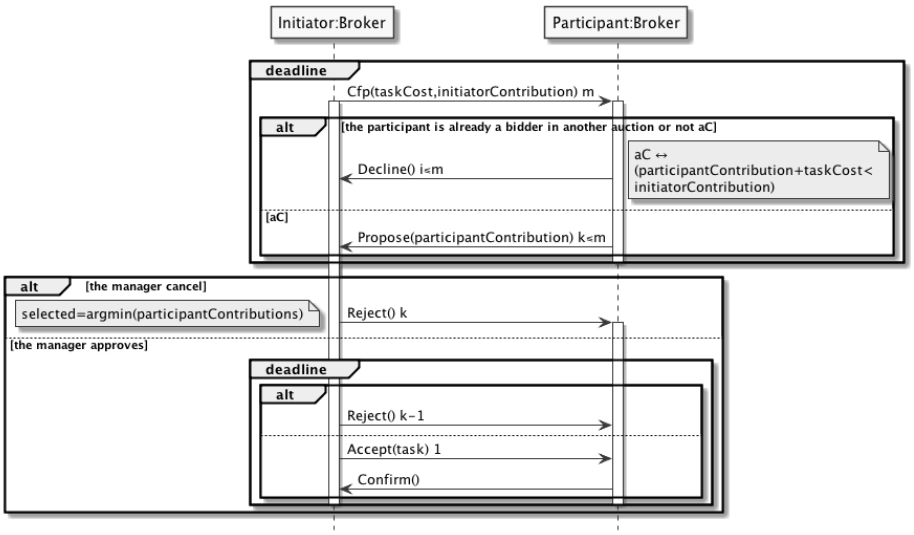


Fig. 2: Negotiation protocol

3.4 Behaviours

Manager. The manager coordinates the activities of the worker and the broker: it provides some tasks to the worker and it triggers the broker to emit `Cfp`. This coordination is based on some principles: i) the manager gives priority to the worker: a task is delegated only if the worker is busy. As soon as the worker is free, the manager gives a new task to it; ii) the manager ensures that a broker is

involved in at most one `Cfp`; iii) the task bundle operates as a priority queue: the manager gives to the worker the task with the highest cost, and tries to delegate the task with the lowest cost via the broker. This task bundle is filled initially by the mappers, and by the broker when it accepts a `Cfp`.

Additionally, the manager interacts with the supervisor to detect the termination of the data processing. The manager is idle when both the worker and the broker are free and the task bundle is empty. The manager is reactivated when it receives a `Request` for the broker.

Worker. The worker, which is initially free, becomes busy as soon as it receives a `Request`. When the task has been performed, the worker informs the manager and it becomes free. During its work, a worker can tell to its manager the estimated remaining cost of the work in progress

Broker. The broker can act as a bidder or as an initiator in a negotiation.

Broker as a bidder. When the broker receives a `Cfp`, it queries the local contribution to the manager in order to participate in the auction. If the acceptability criteria, denoted aC (cf Def. 2), is fullfilled, then a proposal is sent. If it is not the case, then the broker declines to enter the auction and informs the manager that it is free. Since a broker can be a bidder only in one negotiation at a time, it does not reply to all the other `Cfp` but stores them in order to respond as soon as possible. This mechanism prevents livelocks. When the bidder is informed (or not) by the initiator of the negotiation outcome: i) either the bidder wins the auction and it requests the task to its manager and confirms the task delegation to the initiator; ii) either the bidder does not win the auction (the deadline is reached or `Reject` is received) and it informs its manager it becomes free.

Broker as an initiator. When the broker receives a `Submit` from the manager, it sends a `Cfp` to the other brokers. Each reply, whether it is a proposal or declination, is notified in a map. When all of them are received (or the deadline is reached), the best proposal (with the lowest contribution) is selected. Obviously, if no proposal is received the negotiation is cancelled and the broker sends a `Deny` message to the manager. Otherwise, the broker selects the auction winner and rejects the losers. It notifies (`Ready`) the manager that is has found a supplier. In return the manager tells if the task is still available or not, resp. `Approve` or `Cancel`. If it is not the case the negotiation is canceled and the winning bid is rejected. Otherwise, an acceptance is sent and a confirmation is expected.

Halting cfp. When a reducer receives `decline` messages from all its acquaintances in response to its `Cfp`, it is useless to emit again the `Cfp` if the context does not change. The reducer can then enter in a *paused state* to prevent sending useless `Cfp`. In this state the reducer can still respond to other's `Cfp`. It leaves this state only if the context changes. The handling of this state is not detailed here due to lack of space. But the principle is the following: the context change means that an unfulfilled acceptability criteria can become fulfilled.

The only possibilities are: (i) one new task is added to the bundle and then the reducer's contribution increases; (ii) the reducer is informed that some acquaintance's contribution has decreased (one of its acquaintances has delegated a task or its worker has done a task). Even if one of these events occurs, there is no guarantee that the satisfiability criteria becomes satisfied. However the reducer can estimate whether there is a chance this happens since it keeps track of its acquaintances contributions. This is done by storing (and updating) information on contributions received through acquaintances `Cfp`. Thus, the reducer can estimate the chance for the satisfiability criteria to become fulfilled by one of its acquaintance and therefore for its `Cfp` to be successful. When it is the case, the reducer leaves the *paused state*. In the next section, Theo. 3 tells that after a finite number of negotations, every agent will be in paused state. Theo. 4 tells that this happens only when no task transfer could produce a better task partitioning.

With workers accomplishing their tasks, the context changes and some new task transfer could be possible. For instance, this can be the case if one of the worker works more slowly that expected. In this case, agents will un-pause and begin a new negotiation phase that will produce a better new task distribution, i.e. a distribution for finishing the job earlier.

3.5 Results

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

Property 1 *The variance of the reducers contributions decreases after one successful negotiation.*

Proof 1 *Let $\Omega = \{1, \ldots, n\}$ be a set of n reducer agents. Let us consider a successful negotiation led by the agent 1. We denote:*

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

Let $c > 0$ be the cost of the negotiated task and k the reducer agent which has won the negotiation. Due to the acceptability criteria of the participant k, $c_k + c < c_1$, so $c_k + c - c_1 < 0$. Then $Var' - Var < 0$.

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

Property 2 *The successful iterated negotiations make the variance of the contributions decrease.*

[1] It is worth noticing that the negotiation is conservative.

Proof 2 *The protocols ensure that no agent is simultaneously involved in several negotiations: bidder and initiator roles are mutually exclusive for the broker agent; when committed as bidder in negotiation the broker agent does not reply to the requests for other negotiations.*
It results that every negotiation is independent. Then the outcome of a negotiation does not impact another one. According to Theorem 1 every successful negotiation makes the variance of contributions decrease, independently of other negotiations, then during the iteration of such negotiations the variance decreases.

Finally, the negotiation process terminates.

Property 3 *The iteration of successful negotiations terminates.*

Proof 3 *Acccording to Theorem 2, the variance strictly decreases (and is positive) during iterated successful negotiations, and the number of tasks is finite, then after a finite number of negotiations the variance can no more decrease then successful negotiation are no more possible.*

Negotiation process is correct: when it halts, no other task tranfer could alleviate the most loaded agent.

Property 4 *When iteration of sucessful negotiations terminates, there exists no task transfer that could decrease the most loaded agent constribution.*

Proof 4 *Let agent j be the most loaded and τ be the smallest task of agent j. Let us assume that there exists some agent i, with contribution c_i, such that i accepting the transfer of task τ results in a decrease of the highest contribution. This implies that $c_i + c_\tau < c_j$.*

Then i would have make a proposal to a cfp from j for task τ. This cfp would have been successful which is a contradiction.

4 Experiments

In order to evaluate our proposition, we have reimplemented the classical MapReduce using the default partitioning function and our multiagent system with the Scala programming language and Akka's actor implementation.

We have performed several experiments. In all of them, our adaptive process leads to a better task allocation. Due to the lack of space, we have chosen to present a particular illustrative one. It considers the historical weather data in France available since 1996. It contains more than 3 millions of observations over 62 stations (800 Mo). We aims at counting the number of observations per half degree of temperature (cf Fig. 3).

We consider 10 mappers and 20 fully connected reducers. Fig. 4 shows the contributions of reducers in both cases. On the left, the default partitioning function leads to an unfair distribution of tasks where only a few reducers are

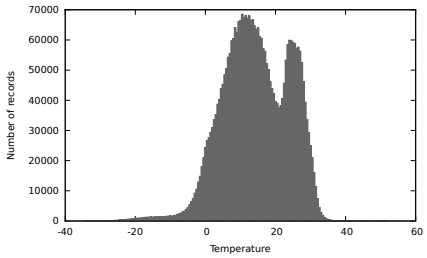


Fig. 3: The number of records per half degree of temperature.

committed to perform tasks. We can observe that the MAS balances the workload between reducers since the tasks are dynamically re-allocated among reducers during the process. The overloaded reducers delegate some of their tasks to unoccupied agents. Actually, the contributions of the worst-off agent is reduced by 72 %, and then the fairness which measures if the processing is performed at expense of the worst-off agent is improved in proportion. Moreover the ratio between the least loaded reducer and the most loaded one shift from 0 to 0.7.

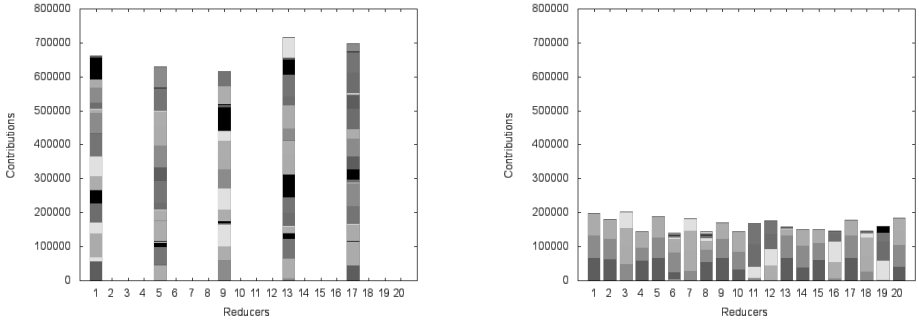


Fig. 4: The contributions of reducers for the classical MapReduce (at left) and for our multi-agent system (at right).

5 Conclusion

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, it is difficult to manage the allocation of work among reducers, since the distribution of tasks is statically fixed. This can lead to an unfair distribution. Our MAS consists of a distributed model of computation, inherently adaptive. Therefore, we have defined in this paper an implementation of MapReduce where task allocation

is the result of negotiations between agents during the reduce phase, with only local decisions taken by reducer agents (i.e. no global supervisor), and without preprocessing the data. More precisely, our model is based on reducers composed of three coordinated agents, manager, worker and broker. In order to balance the workload, these complex reducer agents negotiate tasks based on their individual contributions in order to decrease the contribution of the worst-off agent, i.e. the one which delays the data processing. Our experiments over real-world data confirm that MAS are suitable to design such adaptive allocation.

We consider several perspectives for this work. We are currently distributing the implementation of the framework. Then, we will be able to compare with other skew reduction techniques and measure the communication cost. Another improvement we consider is to split the large key tasks such that the subtasks can be negotiated. Our long-term project consists of tackling complex workflows of jobs and adapting the network of acquaintances to the physical constraints.

References

1. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: Sixth Symposium on Operating System Design and Implementation. (2004) 137–150
2. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association (2012) 15–28
3. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. In: Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07). (2007) 205–220
4. Lama, P., Zhou, X.: Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In: Proceedings of the 9th Internatinal Conference on Autonomic Computing (ICAC'12). (2012) 63–72
5. Verma, A., Cherkasova, L., Campbell, R.H.: Aria: Automatic resource inference and allocation for mapreduce environments. In: Proceedings of the 8th Internatinal Conference on Autonomic Computing (ICAC'11). (2011) 235–244
6. Kwon, Y., Balazinska, M., Howe, B., Rolia, J.: Skewtune: Mitigating skew in mapreduce applications. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD'12. (2012) 25–36
7. Vernica, R., Balmin, A., Beyer, K.S., Ercegovac, V.: Adaptive mapreduce using situation-aware mappers. In: Proceedings of the 15th International Conference on Extending Database Technology, EDBT'12. (2012) 420–431
8. Brandt, F., Conitzer, V., Endriss, U.: Computational Social Choice. In: Multiagent Systems. MIT Press (2013) 213–380
9. Pujol-Gonzalez, M., Cerquides, J., Meseguer, P., Rodríguez-Aguilar, J.A., Tambe, M.: Engineering the decentralized coordination of uavs with limited communication range. In: Proc. of CAEPIA. LNAI 8109 (2013) 199–208
10. Hewitt, C., Bishop, P., Steiger, R.: A universal modular actor formalism for artificial intelligence. In: Proceedings of the 3rd International Joint Conference on Artificial Intelligence. (1973) 235–245