A Coordination Mechanism to Replicate Large-Scale Multi-Agent Systems

Sylvain Ductor
Universidade Federal do Cearà
Fortaleza, Brazil
sylvain.ductor@uece.br

Zahia Guessoum
LIP6, Sorbonne Université, Paris, France
CReSTIC, Université de Reims Champagne Ardennes
Reims, France
zahia.guessoum@univ-reims.fr

ABSTRACT

Distributed cooperative applications are now increasingly designed as Multi-Agent Systems (MAS). Such applications may be open, dynamic, large scale and exhibit heterogeneous and dynamic agents criticalities. These characteristics create new challenges to the traditional approaches of fault-tolerance. We focus on replication-based preventive approach. The aim is to dynamically and automatically adapt the agent replication strategy (number of replicas and their location), in order to maximize the MAS reliability (guarantee of continuity of computation w.r.t the agents criticalities). In this paper, we describe a decentralized coordination mechanism supporting adaptive replication. In order to provide a clear understanding of the specifics of our proposal, we provide a theoretical validation and report on experimental validations by confronting our proposal with an existing solution that is structurally similar.

CCS CONCEPTS

• **Computing methodologies** → **Multi-agent systems**; **Cooperation and coordination**; **Massively parallel algorithms**; • **Hardware** → **Redundancy**;

ACM Reference Format:
Sylvain Ductor and Zahia Guessoum. 2018. A Coordination Mechanism to Replicate Large-Scale Multi-Agent Systems. In *SEAMS '18: SEAMS '18: 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194133.3194154

1 INTRODUCTION

MAS are particularly attractive for creating software that operate in distributed and open environments, such as the Internet. Being both decentralized and self-organized, those systems consist of autonomous entities called agents that are designed to solve problems by cooperating with each other. MAS thus suffer from all the problems associated with building traditional distributed systems as well as the additional difficulties that arise from having flexible and sophisticated interaction among autonomous and adaptive entities. Fault tolerance is one of the most important issues. The design of a fault-tolerant infrastructure that could ensure the continuity of processing even in presence of failures is thus crucial to large-scale MAS.

Failures are traditionally classified as either crash, omission, timing or arbitrary (*i.e. byzantine*) [6]. Two main approaches are used to deal with them: the corrective ones and the preventive ones [14]. In this work, we focus on a preventive approach, the replication. Replication of data and/or computation has been proved to be an effective approach to achieve fault-tolerance in distributed systems [12]. It specifically focuses on crash failures and is not relevant to omission, timing or arbitrary failures. However, it offers the advantage of being application-independent since it does not rely on an application-specific mechanism to discriminate normal from faulty behavior. In our context, we consider a set of agents that run and ought to be replicated on a set of hosts. Hosts may crash, resulting in the lost of every replicas hosted, while a hosted replica consumes a part of the limited amount of resources offered by the host. A replication strategy defines the number and location of each replica of an agent. To be effective, such strategy should level, for each agent, the probability that it will not fail (its *availability*) with its relative importance within the system, its *criticality*, *i.e* the impact that the agent failure would have on the whole MAS (see [11, 14, 15] for details).

Many solutions have been developed (algorithms, architectures and platforms) (e.g. [11, 14, 15]) and many toolkits (e.g. [11, 12, 20]) include replication facilities to build reliable distributed applications. In those approaches, the set of components is supposed to be constant, static and relatively small. The replication strategy may, thus, be chosen by the designer before the application starts. Those approaches are not suited for general MAS contexts where agents can be adaptive and dynamically self-reorganize. In such contexts, criticalities can vary during execution and agents may dynamically join or leave the system. So, *a priori* designed solutions are not suitable. Moreover, in the case of a large-scale MAS, it is crucial to limit the number of replicas of the agents in order to avoid a degradation of the MAS performance. Hence, a strategy which does not dynamically balance the system reliability with the resource consumption would be inefficient or even inapplicable.

In this paper, we introduce a coordination mechanism to manage the replication strategy in large-scale, adaptive and open MAS. This mechanism aims optimizing the MAS reliability *i.e.* replicating agents according to their criticalities. Moreover, the mechanism allows a fine-control of the resource consumption since it can be tuned to bound the host overload, preserving thus the overall MAS performance.

SEAMS '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5715-9/18/05…$15.00
https://doi.org/10.1145/3194133.3194154

This paper is organized as follows. Section 2 describes the related work. Section 3 introduces the problem. The coordination mechanism and its decision processes are described in Section 4 and are theoretically analyzed in Section 5. Finally, Section 6 reports on the implementation and the experiments and Section 7 concludes.

2 CONTEXT & RELATED WORK

Several works study the design of a replication strategy for a MAS [2, 8, 13, 18].

Kraus et al. [18] proposed a probabilistic model for fault-tolerance of a MAS. Zhang et al. [21] extended that model, they proposed two algorithms and five heuristics in order to accurately compute the probability of survival of a given deployment. Even if they consider distributed agents, the proposed algorithms are centralized and thus not scalable. Each heuristic is only efficient for a particular type of instances and none of their solutions considers heterogeneous criticalities and open systems.

Almeida et al. [8], Guessoum et al. [13] and Bora et al. [2] proposed solutions to replicate agents according to their criticality. One grounding hypothesis is that all replicas require the same amount of computational resources. However, in real-life applications, agent resources are often heterogeneous. Also, those approaches are not incremental: any change in the MAS or hosts require to compute again a new solution from scratch.

All those a try to solve this complex problem as a whole. So, they are not suitable to large scale, dynamic and open MAS. We thus propose a decentralized approach, modeled through game theory: each agent has a localized, self-interested behavior that depends on its current state and its current set of neighbors. It evaluates the local allocation from its point of view and tries to optimize its local allocation w.r.t. its preferences. In order to compute efficient replication strategy, our approach requires to guarantee that local improvements always result in a global improvements. To do so, we follow Welfare Engineering design ([3, 4, 9]) by defining the MAS reliability as a decentralizable composition of the agents reliabilities, and then proposing a decentralized coordination protocol which leads agents to implement global improvements from local computations.

3 PROBLEM STATEMENT

A replication-based fault-tolerant MAS consists of a set of agents, $\mathcal{A}gents$, each one associated to a dynamic sets of replicas running on a set of hosts, $\mathcal{H}osts$. Hence, at time t, it can modeled by a bipartite graph:

$$(\mathcal{A}gents(t), \mathcal{H}osts(t), \mathcal{A}llocation(t)) \tag{1}$$

where the current allocation, $A(t) \in \mathcal{A}llocation(t)$, is a binary relation between agents and hosts: each vertex $(a, h) \in A(t)$ represents the replica of Agent a on Host h at time t.

When a host fails, all the hosted replicas are lost. Agent a fails, if and only if, all its replicas are lost. Let $\mathcal{R}eplicas_a(A)$ be the set of hosts where Agent a is replicated in Allocation A and $P(\mathcal{R}eplicas_a(A))$ be the probability that those hosts fail simultaneously. The Agent availability ($\mathcal{D}_a(A)$) is defined as follows:

$$\mathcal{D}_a(A) = 1 - P(\mathcal{R}eplicas_a(A)) \tag{2}$$

Please note that more than one replica of a given agent on a given host does not increase its availability.

The objective of our replication mechanism is to dynamically determine and update efficient and valid allocations that optimize the agents' availabilities with regard to their criticalities (*i.e.* their reliability, see Section 3.1) and that do not overload the hosts (see Section 3.2). An allocation is said to be *valid* if and only if no agent has a null availability (each agent has at least one replica) and no host is overloaded.

Considered MAS are open, large-scale and agents' criticalities and hosts' availabilities may vary dynamically. Each agent criticality, $W_a(t)$, is given and updated by an external component (see for instance [11, 14, 15]).

3.1 Agent & MAS reliability

In this article we exploit three of the most famous social choice functions[1, 5, 7] to define MAS reliability as a function of agents reliabilities. Each function can be computed in a decentralized way: (1) an agent only requires to know its own state and information about the states of its neighbors to compute the local improvement that would result from a reallocation and (2) a reallocation that is locally improving is globally improving.

The *utilitarian welfare* is defined as the expected availability of the system weighted by the agents criticalities. It promotes an overall reliability of the system even if the less critical agents may be likely to fail. The *egalitarian welfare*, on the contrary, focus on securing each and every agent. The egalitarian reliability of an agent is computed by dividing its availability with its criticality[1]. The egalitarian reliability of the society is equal to the reliability of the less reliable agent.The *Nash welfare* encodes a compromise between equality and utility. It is computed as the product of utilities and is not sensitive to agent criticalities by property.

The MAS reliability for a given allocation A at an instant t, is defined as follows[2]:

$$\mathcal{R}_{\text{MAS}}(A)(t) = \begin{cases} \sum_{a \in \mathcal{A}gents} W_a(t) \times D_a(A) & (utilitarian) \\ \min_{a \in \mathcal{A}gents} \frac{D_a(A)}{W_a(t)} & (egalitarian) \\ \prod_{a \in \mathcal{A}gents} D_a(A) & (Nash) \end{cases} \tag{3}$$

where $W_a(t)$ belongs to $]0, 1]$, and $D_a(A)$ belongs to $[0, 1]$.

3.2 Host Load

The host load depends on the local resources used by the agents. In this work, we focus on two important resources: the processor and the volatile memory. Let us note $Proc_h(A)$ and $Mem_h(A)$ respectively, the processor use rate and the memory use rate for Host h in allocation A. The host performances depend on both these parameters; any allocation that overloads at least one parameter is not valid.

Let α_h be a constant parameter that belongs to $[0, 1]$ defined by the designer for host h.
Let $\mathcal{L}_h(A) = \max(Mem_h(A), Proc_h(A))$

$$h \text{ is overloaded} \Leftrightarrow \mathcal{L}_h(A) > \alpha_h \tag{4}$$

[1] the more critical an agent, the lesser its reliability
[2] whenever there is no ambiguity, we omit the t parameter for the sake of clarity

3.3 Optimization Problem

The problem at stake is to maximize the MAS reliability without overloading the hosts and preserving each agent.

$$\begin{array}{ll} argmax_{A \in A} & \mathcal{R}_{\text{MAS}}(A)(t) \\ s.t. & \forall h \in \mathcal{H}osts, \mathcal{L}_h(A) \leq \alpha_h \\ & \forall a \in \mathcal{A}gents, \mathcal{R}eplicas_a(A) \neq \emptyset \end{array} \quad (5)$$

4 COORDINATION MECHANISM

We propose an interaction protocol where agents negotiate atomic reallocations of replicas in order to collectively identify and implement locally improving complex reallocations. An atomic reallocation is implemented as a *Bilateral Contract*; it is a modification of one edge of the allocation bipartite graph. The considered complex reallocations are *host Contracts*, it is to say an update of the replicas allocation of a given host.

Definition 4.1 (Bilateral contract). Let A be the current allocation. A bilateral contract $\mathscr{C}$ is a couple $(a, h) \in \mathcal{A}gents \times \mathcal{H}osts$. It represents an atomic update of the considered allocation that either allocates Host h to Agent a if it is not already allocated or deallocates Agent a from Host h otherwise.

Definition 4.2 (Host contract). Let A be the current allocation. For host h, a host contract $\mathscr{C}_h$ is a set of bilateral contracts, that all concern h. It is associated to a gain, computed as the difference of the social utility of the resulting and the current allocations.

Before initiating the protocol, each agent requests as many replications as possible by sending bilateral contracts to known hosts. The hosts carry out a fast exploration of received bilateral contracts to replicate the largest not-overloading set. This results in a sub-optimal allocation: no more replication can be done but a better allocation may be found by reallocating some replicas. The main issue is then to determine in a decentralized and incremental way which replicas to deallocate and which new replicas to allocate in order to reach a better allocation.

To deal with this issue, each host detects locally improving reallocations by exploring its set of contracts. It includes in this search the set of currently replicated agents (that would be deallocated), and the set of agents that request a replication (that would be allocated). Every host contract that would improves $\mathcal{R}_{\text{MAS}}$ is saved and associated to its gain. The host then sends a bilateral contract of deallocation to each agent that appears in at least one of its improving host contracts and attaches to the bilateral request the highest gain that this agent deallocation can generate. An agent accepts deallocation requests by decreasing social gain; it does not consider its individual interest. This allows to efficiently coordinate the agents' actions without direct interaction.

Figure 1 presents a more detailed description of the protocol. Its steps are:

A-1. Application Each agent sends allocation *proposal* messages to a subset of hosts where it is not already replicated.

H-1. Immediate confirmation Each host receives a set of requests. It randomly[3] selects a maximal subset of agents whose replication would not overload it. It then confirms the allocations to those agents, replicates them, and puts the other applicants in a waiting list.

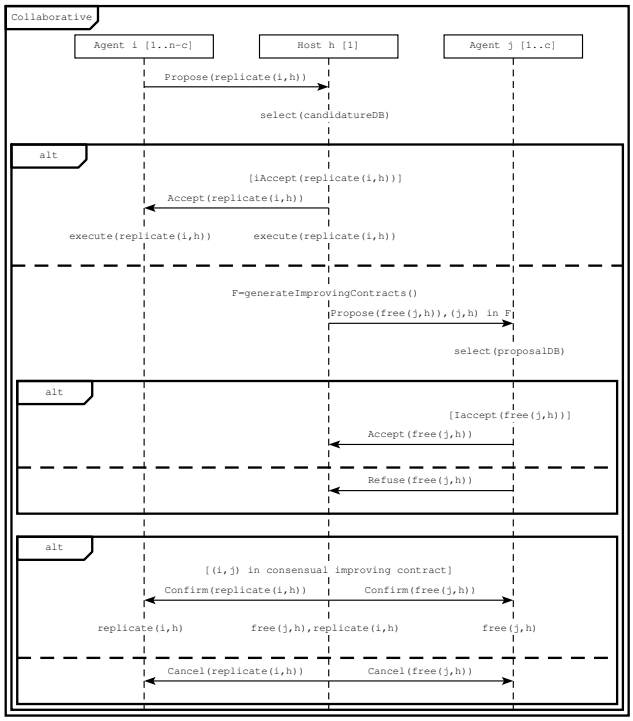


Figure 1: Collaborative protocol for a host h that has replicated c agents, an unallocated agent i and a replicated agent j

H-2. Deallocation Proposal Applicants on the waiting list have a second chance: they may belong to a host contract which would improve the current allocation.

Each host is endowed with a solver that computes all improving reallocation contracts for a considered subset of agents, it is to say the already replicated agents and the agents on the waiting list. The search for improving contracts can be formalized as a bit vector optimization problem. A solution associates to each agent a position. If the corresponding bit equals to "1", it means that the associated agent should be allocated and "0" means that it should not. Each local solution is evaluated thanks to Definition 3. From the host perspective, a valid solution does not overload it and strictly increases its current local social value.

After determining a set of valid solutions, the host sends deallocation *proposal* messages to every agent that is allocated and should not be, according to at least one of the solutions. Each request includes the social value of the best host contract this request could satisfy.

A-2. Acceptation of deallocation Some of the already allocated agents receive a set of deallocation *proposal* messages. They accept them by decreasing social value as long as their constraints (keeping at least one replica) are respected. They reject the others.

[3] we use the roulette wheel algorithm on criticalities

H-3. Confirmation/Cancellation Each host receives either an *accept* or a $reject$ message, for each deallocation *proposal*. The rejects prune its set of multilateral improving contracts: for each rejecting agent, all the multilateral contracts that required this agent deallocation are no longer possible; they are thus removed from the host base. The best remaining host contract is confirmed. $confirm$ messages are sent to the involved agents and *cancel* messages to the others. The host re-initiates the protocol.

A-3. Application Each agent applies the received confirms and re-initiates the protocol.

5 THEORETICAL ANALYSIS

This section presents a complexity analysis of the coordination mechanism and obtained convergence results.

5.1 Exploration in Large-Scale Context & Complexity Analysis

In order to allow the handling of large-scale instances, we bounds the local view of an agent by two parameters: k^{com} bounds the number of neighbors considered when sending a message (*i.e.* the number of simultaneous *proposals*), k^{proc} bounds the number of neighbors considered when carrying out a computation (*i.e.* the number of considered agents while searching for improving host contracts). Those parameters have an impact on the convergence but allow to control the complexity cost depending on the available computation and communication resources.

Regarding communication, the number of messages sent, in one run of the protocol, by an agent or a host is bounded by $2 \times (k^{com} + k^{proc})$. The communication complexity of an agent is thus constant. Indeed, in the allocation protocol, each agent sends up to k^{com} *proposal* and then receives k^{com} $confirm$ or *cancel* messages, and each host sends up to k^{proc} deallocation proposals and then receives k^{proc} $acceptation$ or $rejection$ messages.

Regarding the computation, the most complex algorithm is the host solver that considers a set of k^{proc} agents, with at least one already allocated and another not allocated. The complexity of the solver is function of this constant parameter and is thus constant.

Hence, the complexity of our solution is not affected by the size of the instance and, thus, is compliant to large-scale contexts

5.2 Convergence Analysis

In this section, we present an analysis of the convergence quality of our protocol, where communication and computation are bounded by the k^{com} and the k^{proc} variables introduced in Section 5.1. One execution of the protocol results in an improving reallocation of n' agents on m' hosts. This reallocation can be decomposed into m' independent reallocations of n' agents since hosts do not interact. Agents coordinate by using a shared value: the social evaluation of the submitted deallocation request; accepting them by decreasing value ensures the mutual coherence of the agent decisions. This mechanism results in an iterative acceptation of the feasible hosts contracts by decreasing social value. Our protocol allows thus to explore efficiently and in parallel the reallocation of a subset of k^{proc} agents on each host.

Theorem 5.1 (Convergence). *The system converges and self-stabilizes at infinity in a sub-optimal equilibrium where no host can generate a better allocation by changing its allocation status for at most k^{proc} agents of the system.*

We prove the theorem by proving two lemmas:

(Lemma 1) If there exists an improving contract involving one host and less than k^{proc} agents, this contract is eventually proposed.

(Lemma 2) If at least a feasible improving contract involving one host and less than k^{proc} agents is proposed, a feasible improving contract is selected.

Proof. (*Lemma 1*) Let A be the current allocation and δ a feasible improving contract including a resource, h, and less than k^{proc} agents. The non allocated agents of δ may send an allocation proposal to h and h may select the k^{proc} agents of δ to find improving contracts. Since the solver enumerates all improving contracts for a subset of agents, it will detect δ. δ will be proposed according to the protocol. □

Proof. (*Lemma 2*) A set of one host and less than k^{proc} agents improving contracts are being negotiated in the network. Agents accept the contracts using the same convention: among the ones feasible for them, the ones associated to the best improvement first. Thus, if several contracts feasible for each of their participants have been proposed, at least the one with the best improvement is consensually accepted. □

6 IMPLEMENTATION AND EXPERIMENTS

Although our approach draws its inspiration from welfare engineering [3, 4, 9], it is structurally very similar to the area of Region-Optimal Distributed Constraint Optimization Problems (RO-DCOP) [10, 16, 17, 19], and in particular k-size RO-DCOP.

k-size RO-DCOP solvers are a sub-category of DCOP solvers which proceed by defining each group of k agents as a region and distributing them among the agents, that act as region leaders. Each leader aims to implement the optimal instantiation for the variables of its group members. Hence k-size RO-DCOP guarantees, as we do, that it is eventually not possible to find a better solution by instantiating again simultaneously the variables of at most k agents.

We argue that the originality of our approach *w.r.t* RO-DCOP is that, rather to optimize predefined, static and isolated groups, we focus on a dynamic composition of finely intricated groups. Our hypothesis is that it results in transferring the computational complexity of the local solvers into communication complexity between the system components. In other words, with identical computational resources, our approach should produce significantly better solutions than typical region-based algorithm ; however this should come with a non-negligible communication overhead. If this hypothesis is validated, this would mean that our approach is more suited than RO-DCOP to applications where computation is critic whereas communication is not.

In order to study this claim, we run a set of experiments with the exact same local solvers[4]. In each experiment, we confront a trivial

[4] a $\mu + \lambda$ evolutionary algorithm on allocation bit vector

parallel implementation of the local solver[5], the top-of-the art RO-DCOP solution, DALO (See [17] for details), and our proposal. To allow an execution in large-scale and dynamic context, we proposed some improvements to DALO. In our implementation, each leader is able to iterate its associated groups in a determined order and thus analyzes them one after the other, trying to implement an optimization as soon as it is found. Also, our protocol only considers group formed of 1 host and k agents, whereas DCOP protocol considers all the groups of k members, whether they are agents or hosts. Hence our protocol always considers an allocation vector of k^{proc} binary variables whereas the DCOP protocol considers, in the worst case, an allocation matrix of $\frac{(k^{proc})^2}{4}$ binary variables ($\frac{k^{proc}}{2}$ agents and $\frac{k^{proc}}{2}$ hosts). For the sake of simplicity, we always use k^{proc} to designate both parameters. However for both protocols to be comparable only in the way the interaction has been designed and not be affected by the size of the search space, when we set k^{proc} to k, we set DALO regions size to $\sqrt{4k}$.

We carried out several 30 minutes experiments, averaged 10 times, on a cluster of 12 nodes with 2.3GhZ and 20GB of RAM. Each experiment involves 5 hosts known by every agent; each host can only replicate a mean of 30% of agents. No bound has been done on the communication (see Section 5.1)

6.1 Impact of k^{proc} on the performances

In this first series of experiments, we study the impact of k^{proc}, the parameter defining the size of the subproblems to be solved, on the convergence and the complexity. The number of agents is set to 1000 and k^{proc} varies from 5 to 1000. The experiments have been done with the utilitarian social choice function. As shown by the experiments, while our protocol is not significantly affected by this parameter, the DCOP protocol does not succeed to implement solutions for values superior to 100 (see next paragraphs).

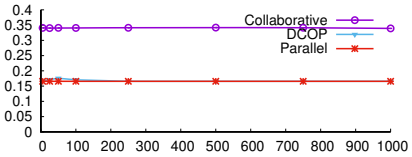
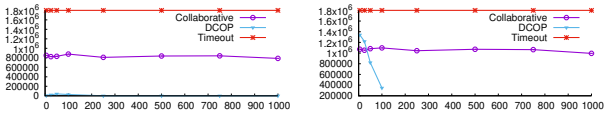


Figure 2: Mean of agents reliabilities as a function of k^{proc}

Figure 2 gives the evolution of the social utility value when k^{proc} varies. We can see that the quality of the optimization provided by our collaborative protocol is the double of the one provided by the DCOP protocol. It is also quasi-constant throughout the experiment: within the allowed time, our protocol is not sensitive to the value of k^{proc}. Please note that the DCOP protocol shows a peak around $k^{proc} = 50$. Indeed this value corresponds to the generation with more groups, and thus, the value where the DCOP protocol exploits the most the distribution.

Figures 3(a) and 3(b) show the individual and social stabilizations. The mean stabilization time is significantly better for the DCOP protocol. However, Figure 3(b) shows the lack of scalability of the

[5]each agent independently generates new individuals that are combined in each new generation

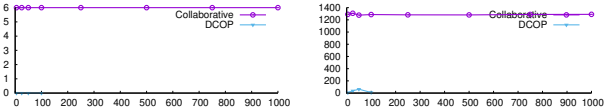


(a) Individual stabilization time (average time in milliseconds)

(b) Social Stabilization Time (maximal time in milliseconds)

Figure 3: Influence of k^{proc} on the stabilization time

DCOP protocol for implementing large instances: DCOP protocol does not succeed to implement new allocations when the k-bound is higher than 100. Meanwhile our solution remains homogeneously resilient throughout the increase of k^{proc}.



(a) Average number of messages sent by an agent

(b) Average number of messages sent by a host

Figure 4: Influence of k^{proc} on the complexity of communication

Figures 4(a) and 4(b) show the average number of messages sent by agents and hosts. In both protocols, hosts send many more messages than agents. This is explained by the fact that there is more than 100 times more agents than hosts.

The collaborative protocol generates many more messages than the DCOP protocol, since it heavily relies on communication as an optimization mean. Collaborative protocol communication complexity appears to be constant w.r.t k^{proc}: agents send an approximate constant number of 6 messages and hosts 1300. This is explained by the fact that k^{proc} only affects hosts' internal computations. DCOP agents and hosts barely send any message: most of the computation is done internally and communication is only used to apply the reallocation.

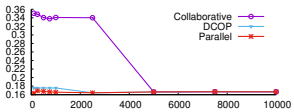
6.2 Protocol performance with varying instance sizes

In this section we analyze the protocol performances with varying instance sizes. k^{proc} is fixed to 100 while the number of agents varies from 100 to 10000.

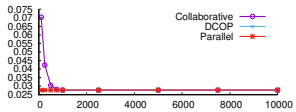
Figure 5 compares the social quality of our protocol with that of the DCOP protocol. On Figure 5(a), averaged reliability is on the ordinate while Figure 5(b) displays minimal reliability and Figure 5(c) the sum of availability logarithms[6]. In all cases, the protocols were configured to optimize the associated welfare.

Both Nash and egalitarian social choice functions are difficult to optimize. The former is a non linear, non concave and non convex function. The latter requires, at each instance, to target one particular agent that cannot be identified in a decentralized way. For those

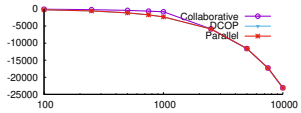
[6]which is the logarithm of the Nash welfare, higher is better - please note that 5(c) uses a logarithm scale



(a) Reliability of the system for the utilitarian welfare



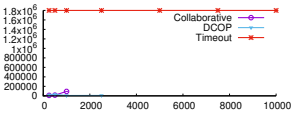
(b) Reliability of the system for the egalitarian welfare



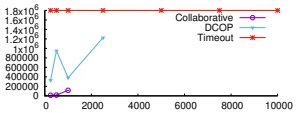
(c) Availility of the system for Nash's welfare

Figure 5: Impact of the number of agents on the reliability and the dynamics of the system

reasons both DCOP protocol and the parallel solver never succeeds in improving the initial allocation. It succeeds in slightly improving the utilitarian social choice function for instances that count less than a thousand ofagents. Our protocol succeeds in improving the allocation when the instance size is below 5000 agents for the utilitarian social choice function, 2500 for the Nash social choice function and 1000 for the egalitarian social choice function. Note that a higher value of k^{proc} or a longer experimentation time may have an impact on the performances of both protocols for the large instances.



(a) Individual stabilization time (average time in ms)



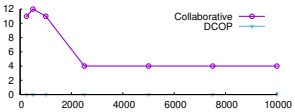
(b) Social stabilization time (maximum time in ms)

Figure 6: Impact of the number of agents on stabilization time

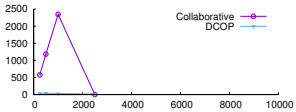
Figure 6 displays the individual and the social stabilization times. The elapsed simulation time was added as a reference. So when the curves intersect it means that the system has not been able to stabilize during the simulation.

The results are consistent with those of the previous figures. The DCOP protocol slowly stabilizes for small instances (whose size is close to k^{proc}), because it is not able to implement a lot of reallocations (for 100 agents, only one host tries to solve the whole problem). The collaborative protocol displays homogeneous stabilization times and always outperforms DCOP protocol in term of social stabilization.

Finally, Figure 7 displays the average number of sent messages per agent (7(a)) and per host (7(b)). The communication complexity of the collaborative protocol is at peek at 1000 agents and drastically reduces at 2500. DCOP communication for coordination remain low throughout the experiments, indeed it requires only few agents to communicate.



(a) Average number of sent messages per agent



(b) Average number of sent messages per host

Figure 7: Impact of the number of agents on the complexity of communication

7 CONCLUSION AND FUTURE WORK

Our objective was to address the problem of an automatic and adaptive management of replication in the context of large-scale, dynamic and open MASs. To cope with scalability, the most constraining of our hypothesis, we have designed a decentralized solution that implements global improvements from local computations. This solution takes inspiration from Micro-Economics [1, 5, 7], Welfare Engineering [4, 9] and Multi-Agent Resource Allocation problem [3].

Our approach is structurally similar to the ones of the area of Region-Optimal Distributed Constraint Optimization Problems (RO-DCOP). Both proceed by implementing local optimizations and provide performance guarantees *w.r.t* the considered region size. However, while RO-DCOP aims to optimize static, predefined and isolated regions, our approach dynamically composes finely intricated regions through interaction. Thus, it produces significantly better solutions, however with a non-negligible communication overhead being, hence, more relevant in context where communication costs are not critical.

Both the theoretical and experimental validations provided a fine understanding of the characteristics of our approach and how it differentiates itself from the literature. However, absolute performance comparison to top of the art solutions have to be done. Also, this work only focused on a theoretical evaluation of our solution, more experiments in presence of fault and dynamic variation ought to be runned. Last, as explained by [19] the local centralization of information of a whole region done by RO-DCOP is a potential threat to privacy. This aspect appears less critical in our solution since agents only communicate through partial, impersonal information. However, further experiments need to be done in order to clarify and evaluate this aspect *w.r.t* the lack of privacy of DALO [17] and the overhead of the solution proposed by [19].

REFERENCES

[1] K. J. Arrow, A. K. Sen, and K. Suzumura. 2002. *Handbook of Social Choice and Welfare*. Vol. Vol. 1. North-Holland.

[2] Sebnem Bora and Oguz Dikenelli. 2006. *Implementing a Multi-agent Organization that Changes Its Fault Tolerance Policy at Run-Time*. Springer Berlin Heidelberg, Berlin, Heidelberg, 153–167. https://doi.org/10.1007/11759683_10

[3] Yann Chevaleyre, Paul E. Dunne, Ulle Endriss, Jérôme Lang, Michel Lemaître, Nicolas Maudet, Julian Padget, Steve Phelps, Juan A. Rodríguez-aguilar, and Paulo Sousa. 2006. Issues in multiagent resource allocation. *Informatica* 30 (2006), 2006.

[4] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet. 2005. Welfare engineering in practice: On the variety of multiagent resource allocation problems. In *Engineering Societies in the Agents World V*. Vol. 3451. Springer, 335–347.

[5] Yann Chevaleyre, Ulle Endriss, Jérôme Lang, and Nicolas Maudet. 2007. A short introduction to computational social choice. In *Proc. 33rd Conference on Current Trends in Theory and Practice of Computer Science*. Springer-Verlag.

[6] F. Cristian. 1991. Understanding fault-tolerant distributed systems. *Commun. ACM* 34, 2 (1991), 56–78.

[7] Bouyssou D., Perny P., and Marchant T. 2009. *Social Choice Theory and Multicriteria Decision Aiding*. Wiley, Chichester -, 868.

[8] Alessandro De Luna Almeida. 2008. *Replication heuristics for agents fault tolerance: a plan-based approach*. Ph.D. Dissertation. Université Paris 6.

[9] U. Endriss and N. Maudet. 2004. Welfare engineering in multiagent systems, In Lecture notes in computer science. *Lecture Note In Computer Science*, 93–106.

[10] Alessandro Farinelli, Alex Rogers, and Nick Jennings. 2009. Bounded Approximate Decentralised Coordination using the Max-Sum Algorithm. In *IJCAI-09 Workshop on Distributed Constraint Reasoning (DCR)*. 46–59. Event Dates: 13th July 2009.

[11] Alan Fedoruk and Ralph Deters. 2002. Improving Fault-tolerance by Replicating Agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2 (AAMAS '02)*. ACM, New York, NY, USA, 737–744. https://doi.org/10.1145/544862.544917

[12] R. Guerraoui and A. Schiper. 1997. Software-based replication for fault tolerance. *Computer* 30, 4 (Apr 1997), 68–74. https://doi.org/10.1109/2.585156

[13] Zahia Guessoum, Jean-Pierre Briot, Noura Faci, and Olivier Marin. 2010. Towards reliable multi-agent systems: An adaptive replication mechanism. *Multiagent and Grid Systems* 6, 1 (2010), 1–24.

[14] Zahia Guessoum, Jean-Pierre Briot, Olivier Marin, Athmane Hamel, and Pierre Sens. 2003. Dynamic and Adaptive Replication for Large-Scale Reliable Multi-agent Systems. In *Software Engineering for Large-Scale Multi-Agent Systems*, Alessandro Garcia, Carlos Lucena, Franco Zambonelli, Andrea Omicini, and Jaelson Castro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 182–198.

[15] Zahia Guessoum, Mikal Ziane, and Nora Faci. 2004. Monitoring and Organizational-Level Adaptation of Multi-Agent Systems.. In *AAMAS*. 514–521.

[16] H. Katagishi and J. P. Pearce. 2007. Distributed DCOP Algorithm for Arbitrary k-Optima with Monotonically Increasing Utility. In *CP 2007 Workshop on Distributed Constraint Reasoning*.

[17] Christopher Kiekintveld, Zhengyu Yin, Atul Kumar, and Milind Tambe. 2010. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1 (AAMAS '10)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 133–140.

[18] S. Kraus, V.S. Subrahmanian, and N. Cihan Tacs. 2003. Probabilistically Survivable MASs. In *IJCAI'03*. 789–795.

[19] Tamir Tassa, Roie Zivan, and Tal Grinshpoun. 2016. Preserving Privacy in Region Optimal DCOP Algorithms. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*. AAAI Press, 496–502. http://dl.acm.org/citation.cfm?id=3060621.3060691

[20] Robbert Van Renesse, Ken Birman, and Silvano Maffeis. 1996. Horus: A Flexible Group Communication System. 39 (04 1996), 76–.

[21] Yingqian Zhang, Efrat Manisterski, Sarit Kraus, V.S. Subrahmanian, and David Peleg. 2009. Computing the fault tolerance of multi-agent deployment. *Artificial Intelligence* 173, 3 (2009), 437 – 465. https://doi.org/10.1016/j.artint.2008.11.007