Installing Resilience in Distributed Constraint Optimization Operated by Physical Multi-Agent Systems

Extended Abstract

Pierre Rust
Orange Labs, France
pierre.rust@orange.com

Gauthier Picard
MINES Saint-Etienne, Laboratoire Hubert Curien UMR CNRS 5516
picard@emse.fr

Fano Ramparany
Orange Labs, France
fano.ramparany@orange.com

ABSTRACT

We study the notion of k-resilient distribution of graph-structured computations supporting agent decisions, over dynamic and physical multi-agent systems. We devise a replication-based self-organizing distributed repair method, namely DRPM[MGM-2], to repair the distribution as to ensure the system still performs collective decisions and remains resilient to upcoming changes. We focus on a particular type of distributed reasoning process to repair, where computations are decision variables and constraints distributed over a set of agents. We experimentally evaluate the performances of our repair method on different topologies of multi-agent systems (uniform or problem-dependent) operating stateless DCOP algorithms (Max-Sum and A-DSA) to solve classical DCOP benchmarks (random graph, graph coloring, Ising model) while agents are disappearing.

KEYWORDS

DCOP; Resilience; Adaptation; MGM-2; A-DSA; Max-Sum

ACM Reference Format:
Pierre Rust, Gauthier Picard, and Fano Ramparany. 2019. Installing Resilience in Distributed Constraint Optimization Operated by Physical Multi-Agent Systems. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 3 pages.

1 INTRODUCTION

We consider the problem of distributing a set of *computations* supporting decisions over a set of agents embodied in physical devices (or *nodes* / agents), like robots, sensors or autonomous cars. Collective and coordinated decisions are organized in a computation graph, where vertices represent computations and edges represent a dependency relation between computations, like in distributed constraint optimization problems (DCOP) [1]. Computations are assigned to on agents according to a distribution function $\mu : X \rightarrow \mathbf{A}$. Here systems must be able to cope with agents additions and failures: when an agent stops responding, other agents in the system must take responsibility and run the orphaned collective computations. As to cope with such dynamics by keeping computations and decisions going whilst the infrastructure changes, one solution inspired by distributed databases is *replication* [5, 6]. In order to ensure resilience of decisions, we thus propose to replicate computation definitions instead of data. More precisely, we define the notion of k-resilience, which characterizes systems able to provide the same functionalities (or make the same decisions) even when up to k nodes disappear, as follows: Given a set of agents $\mathbf{A}$, a set of computations $\mathbf{X}$, and a distribution μ, the system is k-**resilient** if for any $F \subset \mathbf{A}, |F| \leq k$, a new distribution of computations $\mu' : X \rightarrow \mathbf{A} \setminus F$ exists.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.

2 RESILIENCE FRAMEWORK COMPONENTS

Figure 1 summarizes the main components of our reparation framework, called DRPM[MGM-2]. Assuming initial deployment and replica placement have been performed at system boot strap, the system will execute the following repair cycle all along its lifetime: (a) Detect departure/arrival; (b) Activate replicas of missing computations (using MGM-2); (c) Place new replicas for missing computations (using DRPM), and continue nominal operation.

2.1 Computation Distribution

The placement of decision-related *computations* on physical agents has an important impact on the performance characteristics of the global system: some distributions may improve response time, some others may favor communication load and some others may be better for other criteria like QoS or running cost. Assigning computations to agents can be mapped to an optimization problem, as proposed in [4], but remains an NP-hard problem depending on the infrastructure and the computations to distribute. Thus, in a distributed system, solving it each time an agent (dis)appears is not reasonable. We thus propose to repair an initial distribution (that could be optimally computed), using computation replicas.

2.2 Replica Placement

One pre-requisite to k-resilience is to still have access to the definition of every computation after a failure. One approach is to keep k *replicas* (copies of definitions) of each active computation on different agents. Provided that the k replicas are placed on different

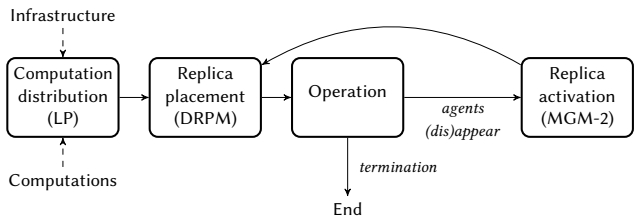


Figure 1: DRPM[MGM-2] life cycle in a glance.

agents, no matter the subset of up to k agents that fails there will always be at least one replica left after the failure, as classically found in distributed database systems [3].

We devise a distributed replica placement method, namely DRPM, to enable each agent to place replicas for its computations on other trustworthy agents. DRPM is a distributed version of iterative lengthening (uniform cost search based on path costs) with minimum path bookkeeping to find the k best paths. The idea is to host replicas on closest neighbors with respect to communication and hosting costs and capacity constraints, by searching in a graph induced by computations dependencies. It outputs a distribution of k replicas (and the path costs to their hosts) with minimum costs over a set of interconnected agents. If it is impossible to place the k replicas, due to memory constraints, DRPM places as much computations as possible and outputs the best resilience level it could achieve. One hosting agent, called *initiator*, *iteratively* asks each of his lowest-cost neighbors, in increasing cost order, until all replicas are placed. Candidate hosts are considered iteratively in increasing order of cost, which is composed of both communication cost (all along the path between the original computation and its replica) and the hosting cost of the agent hosting the replica.

Once all computations have been replicated on k agents, the system can operate. In our experiments, this operation consists in executing DCOP solution methods.

2.3 Decentralized Repair Method

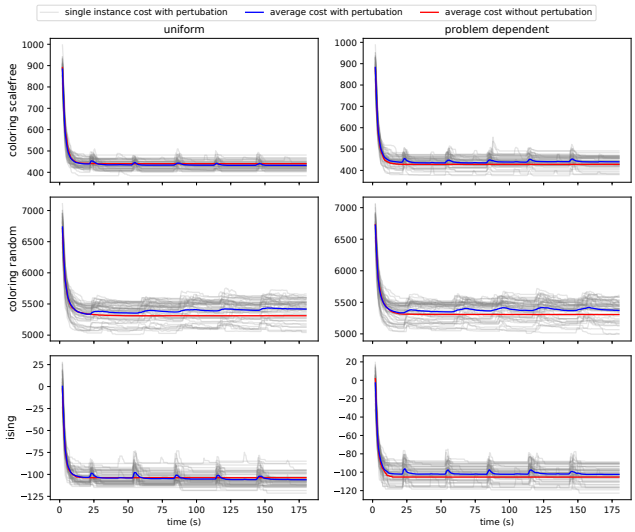
Given a mechanism to replicate computations, we model the repair problem itself as a distributed constraint optimization problem (DCOP) solved using MGM-2 [2], to be implemented by agents following a leave or an entry in the system, as to move some computations to restore the correct function of the system or to increase the quality of the distribution of the computations over agents.

3 EXPERIMENTATIONS

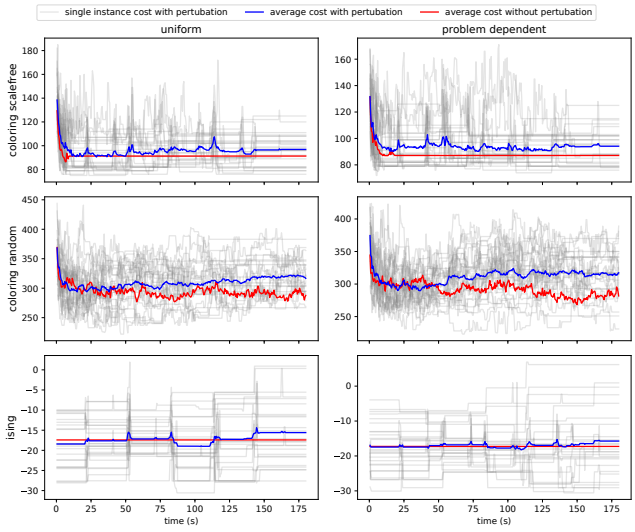
To illustrate our DCOP-based repair framework, we applied to two stateless DCOP solution methods: Max-Sum and A-DSA. Here, factor or constraint graphs are computation graphs to be deployed and repaired at runtime. We run the experiments using the multi-threaded DCOP library pyDCOP[1]. To evaluate our repair framework, we generate benchmarks composed each of three components: a problem definition, a multi-agent topology (the infrastructure), and a disturbance scenario. We study three different types of DCOPs: (i) random graph soft coloring problem, (ii) scale free graph soft coloring, and (iii) Ising problems.

We generate disturbance scenarios as sequences of perturbation events happening every 30 seconds, starting at $t = 20s$. At each such event, k randomly chosen agents disappear as to analyze the impact on DCOP solution methods, and observe the k-resilience of our system.

Figure 2 shows the cost of the solution found by A-DSA and Max-Sum over time, i.e. the performance of the processes to repair. On the different settings we investigate, operating these algorithms is not much impacted by our repair method DRMP[MGM-2], and the systems continue providing solutions, whilst agents are disappearing, which demonstrates the resilience of these systems. Since, our



(a) A-DSA



(b) Max-Sum

Figure 2: Cost of A-DSA (top) and Max-Sum (bottom) solution at runtime, with (blue) and without perturbation (red), on uniform (left) and problem-dependent (right) infrastructure, and when solving scale free graph coloring (top), random graph coloring (middle), and Ising (bottom) problems.

repair method is based on the replication of computations, using problem encoding requiring less computations (choosing constraint graphs instead of factor graphs) is a better choice. Indeed, the complexity of the repair process, encoded as a DCOP itself, strongly depends on the number of computations and the density of the infrastructure. Moreover, on our experiments, Max-Sum operation is much more impacted by agent removals and repairing than A-DSA which is very robust to such dynamics.

[1] https://github.com/Orange-OpenSource/pyDcop

REFERENCES

[1] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. 2008. Decentralised Coordination of Low-power Embedded Devices Using the Max-sum Algorithm. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*. 639–646. http://dl.acm.org/citation.cfm?id=1402298.1402313

[2] R.T. Maheswaran, J.P. Pearce, and M. Tambe. 2004. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS), San Francisco, CA*. 432–439.

[3] G. Malewicz, M. H. Austern, A. J.C Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. ACM, 135–146. https://doi.org/10.1145/1807167.1807184

[4] P. Rust, G. Picard, and F. Ramparany. 2017. On the Deployment of Factor Graph Elements to Operate Max-Sum in Dynamic Ambient Environments. In *8th International Workshop on Optimisation in Multi-Agent Systems (OPTMAS 2017, in conjunction with AAMAS 2017)*. https://www.cs.nmsu.edu/~wyeoh/OPTMAS2017/

[5] R. Wattenhofer. 2015. *Principles of Distributed Computing*.

[6] O. Wolfson and A. Milo. 1991. The Multicast Policy and Its Relationship to Replicated Data Placement. *ACM Trans. Database Syst.* 16, 1 (March 1991), 181–205. https://doi.org/10.1145/103140.103146