Proceedings of the Swarm Intelligence Algorithms and Applications Symposium, Cyrille Bertelle, Gerard H.E. Duchamp and Rawan Ghnemat (Ed.),
at the AISB 2010 convention, 29 March – 1 April 2010, De Montfort University, Leicester, UK

Swarm Intelligence to Distribute Simulations in Computational Ecosystems

Antoine Dutot* and **Damien Olivier*** and **Guilhelm Savin*** †

Abstract. This paper deals with distribution of complex system simulation. A first reflexion is conducted about the distributed environment and we show that it can be considered as a complex system *i.e* an artificial ecosystem, in other words a computational ecosystem. In a second time we propose to manage the complexity of the simulation and its environment by an algorithm based on swarm intelligence. Once again, this proposition is based on interactions and the mechanisms of cooperation and competition that help us to detect the self-organizations which evolve during the simulation in the distributed environment. In a last time, we outline a middleware allowing to distribute dynamically the simulation. This middleware allows mobile code and offers advices to the simulation to reduce the cost of communications. To do that we detect the self-organizations during the simulation and we facilitate their placement on a same node of the distributed system.

1 Introduction

In this paper, we are concerned by complex systems simulation distribution using swarm intelligence. We consider that the simulation evolves in an open distributed environment. We will demonstrate in the following that this environment is a "computational ecosystem". According to this characteristic, we can try to control the trajectory of the system using/encouraging the self-organizations.

This paper is organized as follows. Section 2 defines what is a complex system and our perception of an environment in which complex system simulations can be distributed. Section 3 describes the structure we used to model complex systems. Sections 4 and 5 describes the swarm intelligence algorithm and the platform used to distribute simulations. Finally, section 6 concludes this paper.

2 Computational Ecosystem used to distribute

2.1 What is a complex system ?

A complex system is composed of a massive set of entities with interactions between these entities. A global behavior of the system, which is non-trivial according to entities behaviors, emerges from these interactions. In such system, set of entities is not static: some entities can appear and other disappear: these systems are opened.

Interactions between entities feed an informations flow which structure our system. Furthermore as the system is open, flows cross it bringing the necessary energy to avoid the natural evolution to disorder. We are in front of a kind of dissipative system.

*L.I.T.I.S., University of Le Havre, France
*Authors are sorted alphabetically
†Corresponding author: guilhelm.savin@litislab.fr

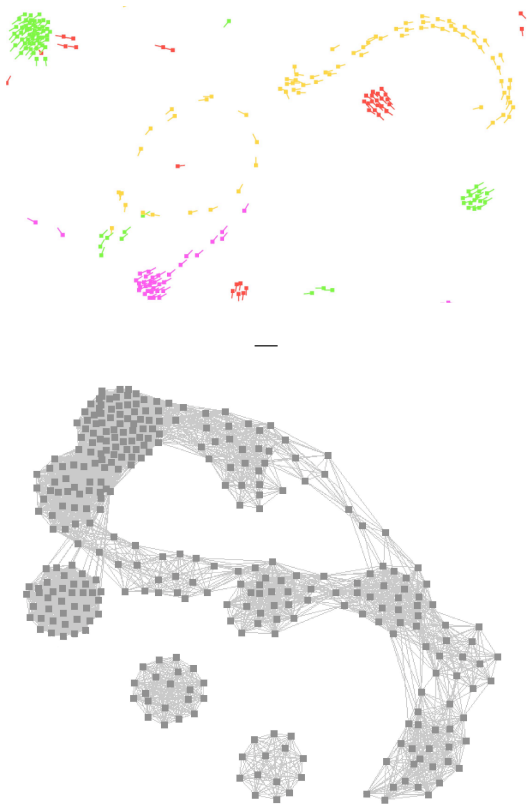

Figure 1. This picture represents one zoom in part of a boids-based simulation (on the top) and its representation with a dynamic graph (on the bottom). On the top part, a snapshot of the simulation is proposed. The boids (interacting entities) can only perceive and interact with fellow creatures located in a limited area. During the process, some groups appear corresponding to global structures obtained from local interactions. On the bottom part of the figure, the graph represents at a given moment the interaction graph corresponding to the state of the system made of boids.

Entity could have preferential interactions between a restricted set of other entities. Entities having such interactions composed a group called organization. There will have some organizations in the system, with a large amount of interactions between organization members, and a few interactions between different organizations.

As organizations are the result of the system dynamic and not a pre-established entry, the mechanism of organizations appearing/disappearing is a mechanism of self-organization. So if it is il-

lusory to try to control each entities to conduct the system, we can plan to affect the system by means of self-organizations.

2.2 Simulation in a computational ecosystem

The environment where the simulations evolve is composed by a set of machines which can connect or disconnect themselves at anytime. On theses machines eventually a lot of executing units (processes, threads, objects, ... process will be used in the following) are running.

There are interactions between these processes which can be direct or not. Direct interactions can be, for example, a communication between two processes or can be done through a shared file or memory zone. Indirect interactions can be a trace drop in the environment like a file. The processes population need resources to *live*: some computing power and some memory at least. Information flow crosses the system as an input data which is consumed by the processes and dissipated as output data. This stream structures the environment.

In the previous description we recognize the definition of a complex system. Furthermore the processes belong to classes as demon, scheduler .... and they are in competition to access to memory for example and collaborate to compute a result for example. There are strong analogies between natural ecosystems and the system that we consider, thus classes can be describe as species. Due to this parallel we define the environment where simulations take place as a computational ecosystem.

The kind of simulation we are interested in is complex system simulations. As we seen above, such simulations are composed of a massive set of entities, needing a large amount of computing resources which can not be provided by a single machine. A way to solve this resources limitation is to distribute the simulation, expanding our computational ecosystem by using a set of machines rather than a single one. The question is now, how to distribute such simulations ?

The elements of answer are in the nature of the simulations and the environment. Finally, environment becomes a complex system that we used to distribute complex systems.

A major raised problem is how to distribute entities in the environment. Load of machines has to be balanced but interactions between remote entities (entities which are located on different machines, these interactions are called *remote interactions*) have to be minimized to reduce network-load. This can be done by trying to control self-organizations. There are a lot of interactions between members of a same organization, if members are located on different machines, amount of remote interactions increases. So, detecting organizations allows to put all members of a same organization on a same machine, reducing remote interactions.

As organizations may change through time, entities location may change too. Problem is that entities are being executed , so they have to stop their execution, store their state and migrate to a new destination. A tool allowing this migration is needed. An interesting concept find in literature is the *active object pattern*[7] which sees the object as an actor receiving requests and executing them one by one. This provides two advantages. First, calls to object methods are seen as requests which allows to model call as an object exportable through the network. A second advantage is that migration becomes trivial: when active object has to migrate, it stops executing its requests, sends them to its next location and starts to execute then on this new location.

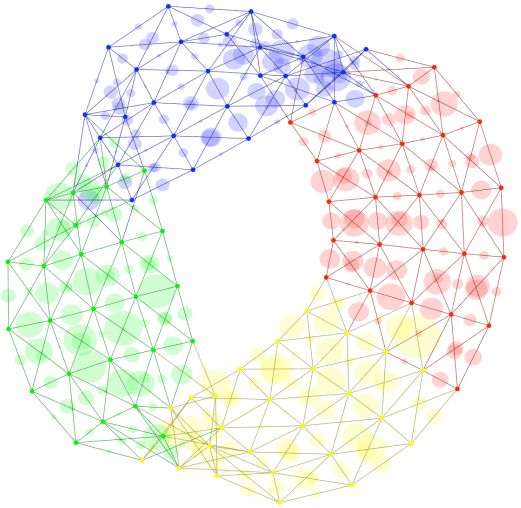

Figure 2. Coloring of a graph representation of the Moebius strip with AntCo2

3 Model

Complex system is composed of a set of entities and there are interactions between these entities. Difficulty of modeling a complex system is that system components may change : entities may appear or disappear and interactions are not static.

The emergence of global properties or behaviors come from this dynamic. Therefore, the study of complex systems leads to three important things to model: entities, interactions and dynamic of the system.

Since interactions can be seen as a couple of entities, a graph can model the system frozen at a time t. A graph G is a peer (V, E) where V is a set of elements called nodes and E is a set of node peer called edges.

To model the dynamic of complex systems, we have to used an other research field which is the one of dynamic graphs. A dynamic graph allows to introduce dynamic in the set of nodes and the set of edges of a graph. A definition of such graphs can be found in [4]. Therefore, a dynamic graph $G(t)$ is a peer $(V(t), E(t))$ where t is a discrete (in this paper) representation of the time. $V(t)$ and $E(t)$ may change with t: $V(t+1)$ (respectively $E(t+1)$) is $V(t)$ (resp. $E(t)$) with eventually some elements added or removed. Each node v and each edge e have a set of properties $P_v(t)$ (resp. $P_e(t)$) which may change with t too.

Figure 1 shows a boids simulation with its corresponding dynamic graph modeling interactions between boids. Boids are a kind of particule introduced by Craig REYNOLDS in [10] to simulate collective animal behavior like birds flocks. If distance between a boid a and a boid b is under a given threshold, then direction of a is influenced by b and inversely: there is an interaction between these two boids modeled in the graph by an edge between nodes representing a and b.

4 Swarm Intelligence - AntCo2

AntCo2 is a distributed algorithm dedicated to load balancing and communication minimisation. It considers only the dynamic graph of the application to compute the distribution. As communications and entities appear and disappear, as the importance of communication

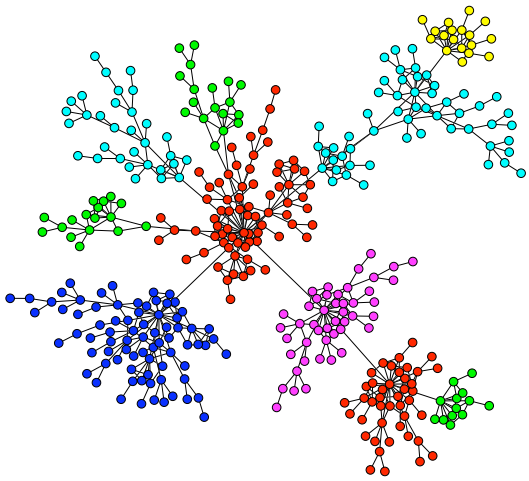

Figure 3. AntCo2 applied to a 400-nodes graph generated using preferential attachment rules.

evolve, the graph changes. Therefore the load balancer should also handle this dynamic process and be able to provide a distribution as the graph evolve.

Each computing resource is associated with a color, then by assigning a color to a node, the algorithm specifies the distribution.

One can see the distribution as a weighted partitioning of the graph. In this partitioning we try to distribute evenly the load (number of entities weighted by their computing demand) and to minimize communications between computing resources to avoid saturating the network. These two criteria are conflicting, therefore a trade-off must be found.

4.1 Detection of organizations

We see the partitioning as a dynamic community detection algorithm. We call such dynamic communities ”organizations”. Communities are often seen as group of vertices that are more densely connected one with another than with the rest of the graph. An algorithm able to detect organizations is able to follow communities as they evolve when nodes and edges appear, evolve and disappear in the communities.

There exists several graph partitioning algorithms ([8, 6, 5]) and community detection algorithms ([9]), but few handle evolving graphs. It is always possible to restart such algorithms each time the graph changes, but this would be computationally intensive. AntCo2 is an incremental algorithm that starts from the previous partitioning to compute a new partitioning when the graph changes.

Having a load balancer running on a single machine, to distribute applications that are often very large could be inefficient. Another goal of AntCo2 is to be able to be distributed with the application.

4.2 Swarm Intelligence

AntCo2 uses an approach based on swarm intelligence, namely colonies of ants. This algorithm provides several advantages: ants can act with only local knowledge of the graph representing the application to distribute. AntCo2 tries to avoid any global computation, therefore allowing it to be distributed with few communications and no global control.

In AntCo2, each colony represents a computing resource and has its own color. Inside colonies, ants collaborate to colonize organizations inside the graph and assign their color to nodes. Inversely, colonies compete to keep and conquer organizations. Figure 3 and 4 show graphs colored by ants.

Ants color nodes using numerical colored pheromones corresponding to their colony color. Such pheromones ”evaporate” and therefore must be maintained constantly by ants. This allows to handle graph dynamics by forgetting old partitioning solutions and discovering new solutions by the constant exploration of ants inside the graph. The details of the algorithm are given in ([2]). Figure 2 shows four ants colonies coloring a graph with pheromones rates on each edge.

The change of a color for a node indicates a ”migration advice”, meaning that the corresponding entity should migrate on the computing resource associated to the new color. An inertia mechanism allows to avoid oscillatory advices.

4.3 Distribution

Ants of the AntCo2 algorithm use only local informations from the dynamic graph. This offers three ways of distribution as described in [3].

A first solution is to have on a single machine a dynamic graph modeling the overall application and to run AntCo2 on this graph. Migration advices are sent to machines. This solution has some disadvantage. If there is a fault on the machine hosting AntCo2, distribution looses its load-balancer: the solution is not fault-tolerant. Moreover, it leads to increase network communications: informations about interactions have to be sent to load-balancer, and migration advices have to be sent to other machines. This is a problem for an algorithm which aims to reduce network-load.

The next solution is similar to the first. Instead of running on a single machine, AntCo2 runs on a set of machines, each machine hosting a part of the graph. This solution becomes more tolerant to fault but still leads to increase network-load.

The last solution is to have one instance of AntCo2 running on each machine. Each instance considers only local entities and interactions: the system becomes decentralized. If there is a fault on a machine, this does not affect the system : this solution is fault-tolerant. As instances using only local informations, there is no need to communicate interactions informations to another machine: network-load not increases. Moreover, computing-load needed by an instance of AntCo2 depends of the amount of entities hosted on the machine, thus distribute entities leads to the distribution AntCo2 itself : AntCo2 is self-distributed.

5 Dagda

Dagda is a middleware dedicated to the distribution of Complex Systems simulations. It uses an existing middleware as a base which is extended with new features. The final aim is to provide a simple way to create distributed complex system simulation.

The main words of Dagda are decentralized, portable, load-balanced. Decentralized means that there is no restricted set of machines on which depend all machines. Dagda aims to be as portable as possible, *ie* any machines (computer,pda,phone,supercalculator...) can participate to the distribution.

Dagda can be divided in three components. The first is a local representation of the part of the application running on the machine. This is model by a dynamic graph. This graph is maintained by the second component of Dagda, called *agency* which is a middleware

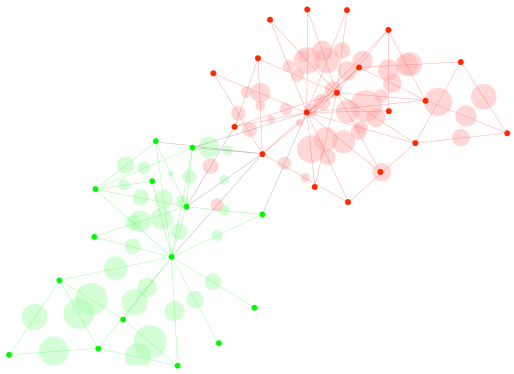

Figure 4. AntCo2 applied to the Zachary Karate Club[11] graph.

with Dagda features. A final aim is to provided a standard connection to middlewares able to manage active object and migration, but actually Dagda allows the use on its internal middleware or the use of ProActive[1], a middleware developed by INRIA. The last component of Dagda is a load-balancing algorithm used to colorize the graph. When node color changes, migration advices are sent to the middleware which proceed to the migration.

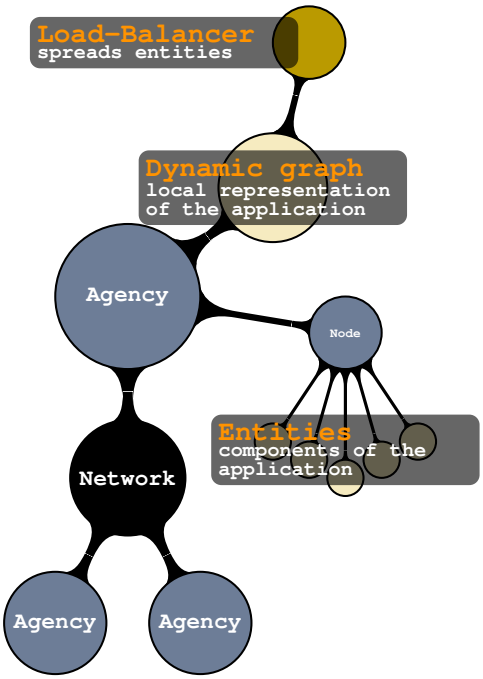

Figure 5. Dagda overview

5.1 Interactions Graph

Dagda is based on the concept that the distributed application is composed of a massive set of objects. These objects are called *entities* and are hosted on a machine by an *agency*. The active object pattern is used to model these entities.

Each machine models its hosted entities in a dynamic graph. Entities are nodes of this graph. When an entity communicates with an other, this is modeled in the graph with an edge between nodes associating to the implicated entities. Greater is the number of interactions between two entities, greater is the weight of the corresponding edge. There is a mechanism which decreases edge's weight through the time.

Entities can migrate from one agency to an other. This raises a problem: how to identify each entity through the network and how to get a remote entity ? The second part of this problem, how to get entity, is treated on section 5.2. Entities are identified by an *id* which is unique through time and network. Uniqueness is assumed by the fact that *id* depends on the agency's address (agency who creates the entity) and on a time-stamp.

Dagda profiles method calls between entities. For example, if an entity `A` calls a method `m()` of an entity `B`, this call will be detected and registered. Then this detection of interactions between entities is used to maintain the dynamic graph which models these interactions through the time.

The GRAPHSTREAM[4][1] API is used to create the graph.

5.2 Agency

The part of Dagda managing remote objects and remote operations is called Agency. This agency is the base of interactions between machines. It allows implementation of the active object pattern by entities and uses a middleware to allow remote method calls and migration of entities. A basic middleware is provided but the aim is to allow the use of any middleware ables to manage active objects.

Dagda aims to have a decentralized architecture so there is no master server to reference informations as for example entities location. Therefore, another aim of agencies is to provide global features without global control.

One is these features is to maintaine a shared context. Overall, environment, composed of all machines, has two kinds of properties: local properties which are specific to a single machine, and global properties which are shared by all machines. Managing local properties is trivial, but managing shared properties without global control raises some problems: if a property is changed on a machine, how spreads this change before other machines look for this property ? Agency aims to solve this problem and provides a valid access to environment properties.

5.3 Load-balancing

The last component of Dagda is a load-balancing algorithm. This algorithm uses the interactions graph and attributes colors to node. Dagda uses the AntCo2 algorithm. This choice allows to :

- balance the work-load of machines;
- reduce the network-load;
- distribute the load-balancer.

Distribution of the load-balancer is an important thing to have a decentralized platform. As describes in 4.3, there are three solutions to run the AntCo2 algorithm. First and second solution use a centralized approach which is not the aim of Dagda. Therefore, the last solution, having one instance of AntCo2 on each machine, has been retained for this middleware.

[1] `http://www.graphstream-project.org`

6 Conclusion

In this paper, an approach of complex system simulation distribution has been presented. This approach considers the execution environment as a computational ecosystem. We have seen that this environment becomes a complex system used to compute complex system simulations.

A major problem raised in this paper is how to distribute entities composing complex system. A swarm intelligence algorithm of organizations detection has been presented and used as load-balancing algorithm. Then a platform dedicated to complex system simulations distribution and using the previous load-balancing algorithm has been presented.

Next steps of this work can be divided in two parts. The first is about the platform, Dagda, which still needs some development and a phase test. Second part is about AntCo2 results and the adaptivity of load-balancing. Some mechanisms need to be added to avoid oscillatory migration advices. Adaptivity means that organizations have to adapt to environment, taking account for example of machine resources, other processes running on the machine…

REFERENCES

[1] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel, and Romain Quilici, *Grid Computing: Software Environments and Tools*, chapter Programming, Deploying, Composing, for the Grid, Springer-Verlag, January 2006.

[2] Cyrille Bertelle, Antoine Dutot, Frédéric Guinand, and Damien Olivier, 'Organization detection for dynamic load balancing in individual-based simulations', *Multi-Agent and Grid Systems*, **3**(1), 42, (2007).

[3] Antoine Dutot, *Distribution Dynamique Adaptative à l'aide de mécanismes d'intelligence collective*, Ph.D. dissertation, Université du Havre - LIH, 2005.

[4] Antoine Dutot, Frédéric Guinand, Damien Olivier, and Yoann Pigné, 'Graphstream: A tool for bridging the gap between complex systems and dynamic graphs', in *EPNACS: Emergent Properties in Natural and Artificial Complex Systems*, (2007).

[5] C. M. Fiduccia and R. M. Mattheyses, 'A linear time heuristic for improving network partitions', in *ACM IEEE Design Automation Conference*, pp. 175–181, (1982).

[6] B. Hendrickson and R. Leland, 'An improved spectral graph partitioning algorithm for mapping parallel computations', *SIAM J. Scien. Comput.*, **16**(2), 452–469, (1995).

[7] Carl Hewitt, Peter Bishop, and Richard Steiger, 'A universal modular actor formalism for artificial intelligence', in *t*, pp. 235–245, (1973).

[8] B.W. Kernighan and S. Lin, 'An efficient heuristic procedure for partitioning graph', *The Bell System Technical Journal*, **49**(2), 192–307, (1970).

[9] M. E. J. Newman and M. Girvan, 'Finding and evaluating community structure in networks', *Phys. Rev*, **69**, (2004).

[10] Craig W. Reynolds, 'Flocks, herds and schools: A distributed behavioral model', in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34, New York, NY, USA, (1987). ACM.

[11] W. W. Zachary, 'An information flow model for conflict and fission in small groups', *Journal of Anthropological Research*, **33**, 452–473, (1977).