Cooperative Mobile Agents to Gather Global Information

Michel Charpentier
Department of Computer Science
University of New Hampshire
Durham, New Hampshire NH 03824, USA
Email: charpov@cs.unh.edu

Gérard Padiou and Philippe Quéinnec
Institut de Recherche en Informatique de Toulouse
ENSEEIHT, 2 rue Camichel, BP 7122,
F-31071 Toulouse cedex, France
Email: {Gerard.Padiou, Philippe.Queinnec}@enseeiht.fr

***Abstract*— This paper describes an original approach to writing reactive algorithms on highly dynamic networks. We propose to use randomly mobile agents to gather global information about such networks. In this model, mobility is twofold: nodes move within the network and agents are able to migrate from node to node by following network links as they exist at a given moment in time. We apply this approach to a toy load balancing example. Through simulations, we illustrate the reactivity and convergence properties of the proposed approach. In particular, we emphasize two points: the solution is well adapted to node mobility insomuch as its performance increases with the node mobility rate, and agent cooperation can be used effectively in this context to increase performance.**

I. Introduction

This article describes our proposed approach to dealing with highly dynamic networks, such as ad-hoc networks or sensor networks. Two observations can be made about these networks. First, they call for a reexamination of some of the assumptions distributed algorithms traditionally rely on: a mostly static network topology and a limited number of sites or link failures. Second, the client-server strategy, based on remote procedure calls and commonly used in many applications, is no longer workable. The notion of a server that can always be reached by clients no matter their location, is incompatible with highly dynamic networks. Instead, truly distributed solutions (both in control and data) are required.

An illustrative instance is the notion, extensively studied, of consistent global states, or snapshots. With the advent of dynamic networks, snapshots may not be such an apposite notion anymore and could even become inappropriate. Because of the instability of the network and its dynamic nature, it is unlikely that any given node will manage to obtain a meaningful and useful snapshot of all (or most) participants. In all probability, a site will have a workable knowledge of the states of its close neighbors, one or two communication hops away, depending on how unstable the network is. Larger snapshots may then be too inaccurate as to be of real value.

In this paper, we advocate a distributed model that relies on cooperating mobile agents to gather global information [1], [2]. Three characteristics of mobile agents, in particular, suggest that they may be well adapted to dynamic networks:

- *fault tolerance*: by having multiple agents that share a common behavior, a system can be made tolerant to the failure of a limited number of agents.
- *adaptability*: the behavior of an agent can change from node to node, based on its current state and the information it has gathered while traveling the network.
- *cooperation*: agents can cooperate by exchanging data and/or code when they meet on a given site.

Therefore, when dealing with dynamic networks, a practicable view of the underlying system can be provided to application-level processes by a layer of lightweight mobile agents randomly and rapidly moving from node to node, without the need for point-to-point communication and (impractical) global routing.

We illustrate our approach in this article through a simple load balancing example. Many load balancing algorithms have been optimized for static or quasi-static networks, with or without the possibility of site failure [3]. Algorithms have also been devised for peer-to-peer systems [4], where networks are more dynamic but the assumption of global routing is maintained [5]. In contrast to such algorithms, our approach is based on weaker assumptions about communication, similar to what is found in ad-hoc networks: a node can only communicate with a limited and continually evolving set of neighbors. Under such a weak assumption, we show how mobile agents can approximate global information about the underlying dynamic system by data aggregation and rely on this knowledge to migrate entities and achieve load balancing.

We model network mobility as random, and our mobile agents also make random decisions as to which nodes they are going to visit next. Such random walks have been proposed as a primary algorithmic principle in protocols addressing searching and topology maintenance of unstructured P2P networks [6], in group communication protocols in ad-hoc networks [7] and for routing in wireless networks [8]. As an example, we describe an algorithm that uses mobile agents to approximate the average load of the network and perform a load balancing task. This algorithm exhibits significant reactive and stabilization properties. In the conclusion, we discuss other possible applications of using such random agents.

The load balancing example illustrates several features of our approach to dealing with dynamic networks. Firstly, agents do not attempt to identify nodes and to build a global picture of the network. Individual agents start with no knowledge of either the total number of other agents or the number and the interconnection of active network nodes, and nodes are anonymous. Furthermore, nodes are not used by agents to store any kind of information. Secondly, mobile agents can be made more "reactive" by adjusting some of their parameters. This allows us to rely on the same approach to deal efficiently with applications that are more or less dynamic. Thirdly, several agents can be used to perform a given task, although each agent need not be aware that other agents are also working on this task. By using several independent agents, an algorithm can be made both more efficient and more fault-tolerant. Finally, agents can collaborate with other agents when they are presented with such an opportunity. This does not mean that agents need to be aware of the existence of other agents, just that they may collaborate when they randomly encounter each other. We show how performance can be improved overall by using such an opportunistic form of cooperation.

The remainder of the paper is organized as follows. We introduce the load balancing problem and the principle to our solution in section II. Section III presents possible improvements to the basic algorithm, as discussed above, namely *reactivity adjustment* (sec. III-A) and *cooperation* (sec. III-B). We compare our approach to related work in section!IV and we conclude in section V.

II. The Load Balancing Problem

A. Problem Description

We consider a multi-entity application running on a dynamic network in which nodes may appear or disappear and links between nodes can be changed. Entities may be active (processes) or passive (files). Entity creation can be internal (a process creating new processes) or external (new files being uploaded). At any time, new entities can be created on a given node. It is the responsibility of the load balancing algorithm to distribute these new entities among reachable nodes.

Entities cannot migrate on their own but, of course, entities can disappear (process exit or file deletion) at any time, hence lowering the load of the host node they were located on. In this context, the task of a load balancing algorithm is to make sure that entities are moved from high load nodes to lower load nodes. Thus, we focus on the so-called continuous and dynamic case of load balancing, in which new entities are generated in place as time proceeds [3].

B. Basic Algorithm

The load balancing task is performed by at least one mobile agent, but there may be several agents running concurrently. These agents travel the network randomly and, on each visited node, they decide to push entities in excess towards other nodes. This decision is based on an agent's understanding of the average load in the network.

The global (actual) average load is defined as the total number of entities divided by the total number of nodes in the dynamic network. Obviously, it is impossible for an agent to know this number exactly, and agents should compute approximations of this average load. Such an approximation is computed as follows:

- Each agent maintains a variable nbHops, which is increased by one every time the agent moves from one node to another node. It should be noted that nbHops is *not* the number of different nodes visited by the agent, but the number of migration steps of this agent, including nodes that are visited multiple times.
- Each agent also uses a variable mean to count the number of entities seen on the network. Every time an agent arrives on a node (where), mean is (re)evaluated according to the number of entities currently on this node ($\mathsf{where.entCnt}$).

Agents approximate the average load by computing a new mean from their current approximation of the mean and the number of entities on the current node:

$$\mathsf{mean} = \frac{(\mathsf{mean} * \mathsf{nbHops} + \mathsf{where.entCnt})}{\mathsf{nbHops} + 1} .$$

An agent moves randomly during at least P steps before using its approximation of the mean to push entities. Afterward, on each visited node, it computes a current approximation mean. Then, if the number of entities currently located on this node is greater than mean, the visiting agent sends a migration order to entities in excess. These entities must leave the current node and move toward a neighbor node. The choice of the destination neighbor is random.

C. Properties of the Algorithm

In its basic form, the algorithm described above enjoys several useful properties:

Load Balancing Property. If a network node is overloaded (i.e., it contains more entities than the actual load average), one of two things will eventually happen: either the actual load average will increase enough for this node not to be overloaded anymore, or one or more entities will eventually leave the node, either from termination (deletion) or from being ordered to migrate by an agent. It should be noted that, unlike other algorithms, this approach requires no global knowledge [9]. In particular, no agent knows either how many nodes exist in the network or how many entities are currently present.

Convergence property. If the underlying application reaches a stable point, i.e., a point where no new entity is ever created, the algorithm enjoys a stronger property: it will reach a state where no node contains more entities than the average load.[1] This convergence property [10] does not require the underlying network to stop being dynamic: nodes remain mobile and may still be completely disconnected during finite periods of time.

Fault Tolerance Property. The algorithm is fault-tolerant [11] in a natural way. The loss of an agent can be recovered by using a set of agents. If an agent is lost or is locked on its current (disconnected) node, others can continue their load balancing task. As long as at least one agent survives, the algorithm still works, albeit with diminished performance.

[1] In case the average load in not an integer value, the stable state is a state where no node contains more entities than the *ceiling* of the average load.

These properties were evaluated experimentally, using a simulation in Java. Furthermore, the load balancing and convergence properties have been verified on small networks using a formal description in TLA+ [12].

III. Reactivity and Cooperation

A. Reactivity Adjustment

If load balancing agents always compute their approximations of the mean starting from the initial state, long executions will result in slow updates of this mean with respect to actual variations in the number of entities. In other words, such variations are less and less quickly perceived by load balancing agents. This is because new visits to nodes have less and less numerical impact on the computation of the mean.

This situation can be problematic if the global number of entities is varying quickly. For instance, imagine a situation where the total number of entities has been relatively stable for some time, but the application suddenly creates a substantial burst of new entities. In the basic version of the algorithm, agents that have been running for a long time will be slow to move these new entities away from high load nodes.

This drawback can be avoided by letting agents perceive the instability of the underlying application through a standard deviation computation. Agents cannot evaluate the actual standard deviation, but they can follow the same approach as with the load and compute successive approximations of the standard deviation:

$$\mathsf{sd}' = \sqrt{\frac{\sum_{k=1}^{k=d}(i_k.\mathsf{entCnt} - \mathsf{mean})^2}{d}} \ ,$$

where mean is the current value of the mean as computed by the agent and d is a predefined walk length.

The behavior of an agent is then the following: if the increase between two successive values of the approximated standard deviation sd' is greater than a chosen threshold, an agent reinitializes its variable nbHops to a low value. This update increases the reactivity of the algorithm by making subsequent node visits having more weight in the computation of the approximate mean.

B. Cooperation

In our model, agents can proceed independently from one another, or they can cooperate. In the load balancing example, cooperation among agents can be used to improve the approximation of the average load for agents that participate in such cooperation steps.

The cooperation protocol is synchronous and relies on a client-server mechanism. When an agent visits a node, its name is recorded in a local name server and the agent enters a cooperating state. In this state, it can either initiate cooperating interactions with other local agents or accept to cooperate with other requesting agents. When a client agent has found an accepting partner, it can call a set of methods provided by this agent. At any moment, an accepting agent may decide to refuse new clients. However, it must wait for an explicit termination order from all its clients before it can leave the node.

The load balancing algorithm can benefit from the cooperation protocol described above to gain in efficiency. When an agent visits a node, it initiates a cooperating step if it finds at least another agent on this node. For agent A, a cooperation step consists of collecting the current state of one or more other agents, and using this information to update A's own current state in the following way:

$$\mathsf{mean} = \frac{\sum_{i=1}^{i=p}(i.\mathsf{mean} * i.\mathsf{nbHops})}{\sum_{i=1}^{i=p} i.\mathsf{nbHops}} \ ,$$

where p is the number of cooperative agents (including the agent that initiated the cooperation step).

Such cooperation steps make it possible to aggregate the partial results of random walks performed by different agents. Insofar as different agents explore different sets of nodes, the aggregation of their estimates results in a better approximation of the global mean.

IV. Related work

Mobile agents have been proposed as a way to solve a variety of distributed problems. In [13], a framework based upon cooperative mobile agents is used to achieve load sharing in distributed web server groups. *Load information agents* are used to gather global information about workload at the participating servers. *Job dispatching agents* are instantiated by overloaded servers to perform the job reallocation to the appropriate remote servers. This framework differs from our approach is several points:

- The main goal is not the same: their goal is the global throughput optimization (increase) of servers in term of executed requests.
- Random walk is not used as a basic mechanism to gather global informations. The number of servers involved in the load sharing task is low in web servers groups.
- Our load balancing agents perform simultaneously and continuously both information gathering and job dispatching tasks.
- No convergence property is studied.

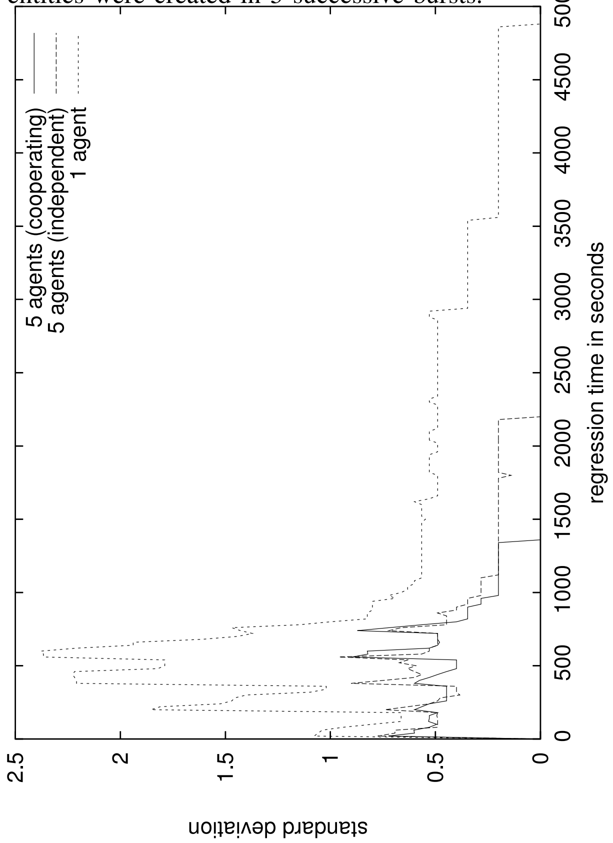
Load balancing in peer-to-peer networks [14] is a basic problem [15]–[17]. Originally, flooding was used in peer-to-peer networks, but random walks have recently been proposed for searching and for maintaining strong connectivity properties in unstructured networks [6]. We think our approach of combining mobile agents and random walks, already used in [8] to achieve routing in wireless networks, could be applied to load balancing in peer-to-peer networks with files or virtual servers (sets of files) as entities.

V. Conclusions and Future Work

Many of today's distributed systems involve a high degree of dynamic evolution and call for reliable adaptable algorithms with good reactivity. For instance, we believe that the approach presented in this paper is well suited to mobile ad-hoc networks or grid computing applications.

Through a load balancing example, we have presented an agent-based approach to dealing with such constraints. We propose to rely on agents that travel a network randomly, work individually most of the time, but also cooperate with other agents when given the opportunity. The resulting algorithm enjoys useful properties, both for continuous load balancing and for reaching a stable state if entities stop being created.

The load-balancing algorithm has been simulated in Java. In particular, we analyze the impact of the number of agents, of their cooperation, of the number and mobility of nodes and of the reactivity adjustment described in section III-A. The reader is referred to the full paper [18] for a comprehensive description of the results. As an example, the figure below shows how the number of agents and their using cooperation or not impacts the performance of the algorithm in a situation where entities were created in 5 successive bursts.



In our model, agents can be initiated without any knowledge of their environment (in particular regarding the network size and connectivity), which gives our approach similarities with self-stabilizing algorithms. Indeed, simulation shows that a nonempty set of agents with no initial knowledge of their environment can compute global information such as an average load and a standard deviation.

This approach can easily be generalized to compute other global parameters. For instance, by computing the average number of migrations it takes to come back to a marked node, agents can estimate the size of a dynamic network and, together with the average load, the total number of entities. By storing different kinds of information on network nodes, agents can compute the global amount of available resources of various kinds (processor power, storage, ...) and regulate network activity more precisely. With such an approximation, random server agents could be used to implement a long-term scheduling strategy: according to the current approximation of the available global resources, the creation of new entities could be allowed or delayed. In another paper, we have used the same agent-based model to implement a fully distributed localization service. Starting from local naming servers, an underlying layer of lightweight mobile agents moving randomly is responsible for spreading gossips [19] about the location of resources or application-level agents. An agent of the application looking for a specific entity can then move from node to node along the path built by the gossip propagation.

We have already formalized basic distributed algorithms in the context of fixed networks [2]. However, the context of randomly evolving networks and agents calls for a different formalization framework. We are currently investigating the relationship between our approach and random walks in (static) graphs [20]. This way, we hope to be able to confirm simulation results with formally stated properties guaranteed by our approach, in particular liveness and stabilization properties.

References

[1] D. Milojicic, F. Douglis, and R. Wheeler, *Mobility : processes, computers and agents*. Addison-Wesley, 1999.

[2] M. Charpentier, M. Filali, P. Mauran, G. Padiou, and P. Quéinnec, "Modelling and Verifying Mobility : A Case Study ," in *3rd Int'l Conf. On Principles Of Distributed Systems , Hanoi, Vietnam*. OPODIS, octobre 1999, pp. 151–166.

[3] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press - John Wiley & Sons, Inc., March 1995.

[4] A. Sohn, R. Biswas, and H. D. Simon, "A dynamic load balancing framework for unstructured adaptive computations on distributed-memory multiprocessors," in *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM Press, 1996, pp. 189–192.

[5] M. Alberto, M. Heing, and B. Özalp, "Messor : Load-balancing through a swarm of autonomous agents," in *Proceedings of the 1st International Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, S.-V. G. Lecture Notes in Computer Science, Ed., vol. 2530, Bologna, Italy, July 2003, pp. 125–137.

[6] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walk in peer-to-peer networks," in *IEEE INFOCOM*, 2004.

[7] S. Dolev, E. Schiller, and J. Welsh, "Random walk for self-stabilizing group communication in ad hoc networks," in *Proceedings of the twenty-first annual symposium on Principles of distributed computing (PODC'02)*. New York, NY, USA: ACM Press, 2002, pp. 259–259.

[8] M. Bui, S. K. Das, A. K. Datta, and D. T. Nguyen, "Randomized mobile agent based routing in wireless networks," *International Journal of Foundations of Computer Science*, vol. 12, no. 3, pp. 365–384, 2001.

[9] P. Berenbrink, T. Friedetzsky, and E. W. Mayr, "Parallel continuous randomized load balancing," in *Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures*, 1998, pp. 192–201.

[10] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM*, vol. 17(11), no. 11, pp. 643–644, 1974.

[11] M. R. Lyu, *Software Fault Tolerance*. John Wiley & sons Ltd, 1995.

[12] L. Lamport, *Specifying Systems : The TLA+ language and tools for Hardware and Software Engineers*. Addison Wesley Professional, 2002.

[13] J. Cao, X. Wang, and S. K. Das, "A framework of using cooperating mobile agents to achieve load sharing in distributed web server groups." *Future Generation Computer Systems*, vol. 20, no. 4, pp. 591–603, 2004.

[14] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications'," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, February 2003.

[15] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, “Load balancing in dynamic structured P2P systems,” in *IEEE INFOCOM*, 2004.
[16] D. R. Karger and M. Ruhl, “Simple efficient load balancing algorithms for peer-to-peer systems,” in *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2004, pp. 36–43.
[17] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, “Load balancing in structured p2p systems,” in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS’03)*, 2003.
[18] M. Charpentier, G. Padiou, P. Quéinnec, and C. Cubat-dit-Cros, “Cooperative mobile agents to gather global information (full paper),” http://www.enseeiht.fr/∼queinnec/NCA05.pdf, May 2005.
[19] A. Demers *et al.*, “Epidemic algorithms for replicated database maintenance,” in *6th Symposium on Principles of Distributed Computing*, Aug. 1987, pp. 1–12.
[20] R. Motwani and P. Raghavan, *Randomized Algorithms*, ser. Cambridge International Series in Parallel Computation. Cambridge University Press, 1995.