Applying Self-Aggregation to Load Balancing: Experimental Results

Elisabetta Di Nitto, Daniel J. Dubois, Raffaela Mirandola
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano, Italy
(dinitto,dubois,mirandola)@elet.polimi.it

Fabrice Saffre, Richard Tateson
BT Group
Adastral Park
Ipswitch IP5 3RE, UK
(fabrice.saffre,richard.tateson)@bt.com

ABSTRACT

One of the today issues in software engineering is to find new effective ways to deal intelligently with the increasing complexity of distributed computing systems. In this context a crucial role is played by the balancing of the work load among all nodes in the system. So far load balancing approaches have been designed for networks with fixed or dynamic topologies. These approaches work well in the case each node knows its similes and is able to contact them to delegate tasks. However, they do not address the needs of more dynamic systems where nodes are able to process different types of jobs and have limited knowledge about their neighbors and the whole system. To address these issue, we are experimenting with the usage of autonomic self-aggregation techniques that rewire such highly dynamic systems in groups of homogeneous nodes that are then able to balance the load among each others. We present our approach and show through simulation that it provides significant advantages under the circumstances described before.

Keywords

Autonomic computing, self-organizing systems, load balancing, clustering, performance analysis

1. INTRODUCTION

One of today issues in software engineering is to find new effective ways to deal intelligently with the increasing complexity of distributed computing systems. In this context a crucial role is played by the balancing of the work load among all nodes in a system.

The literature in the area of load balancing has produced several algorithms aiming at supporting the nodes in understanding when they are overloaded and in deciding if to delegate part of their tasks (see for example [25]). These algorithms are mainly based on the use of iterative methods derived from the linear systems theory [10] and they iteratively balance the load of a node with its neighbors until the whole network is globally balanced. They can be used in networks with fixed topologies or dynamic topologies, but they work well only in the case each node knows its similes and is able to contact them to delegate tasks.

Therefore, there is the need to find new approaches that could work properly in highly dynamic contexts. Think, for example, of a system where nodes enter and exit the system without following any rule (e.g., a conference room where persons are free to participate or a megastore where customers enter and exit continuously). The system is composed of sensors and actuators, associated to people and to service centers, with different capabilities and able to deal with and to react to possible critical situations by dividing the workload among similar nodes (e.g., nodes — representing people — in the conference room willing to share the lecture recording and others — representing specialized clerks — trying to share high numbers of customers with other clerks to be able to fulfill their requests).

To address this issue, we are experimenting with the usage of autonomic self-aggregation techniques that rewire a system composed of heterogeneous nodes in groups of homogeneous nodes that are then able to balance the load among each others using classical techniques.

To this end we take advantage from researches that borrow some ideas from the biological world [27]. For instance, they study the capability of various species to evolve in order to better adapt to the environment they are living in. Also, they analyze the behavior of colonies of insects and their capability to *self-organize* [12, 7]. In this last case, the main goal is to apply similar capabilities to software systems of interconnected components that singularly, like ants for their anthill [2], have limited information and reasoning power, but, all together, contribute to the high-level goals for the whole system. Using this approach many complex problems can be solved by executing simple rules locally to each component of the system, regardless system size and without the need of a centralized control [5].

In this context, *self-aggregation* algorithms aim at establishing and maintaining groups of components that cooperate to reach a common goal. The applications of these algorithms include all cases in which there is a need for continuously reconfiguring those groups. Besides the pervasive computing examples we have mentioned before, think also at the case of a network of message brokers that need to be restructured because of a failure in one of its portions.

In this paper we experiment with the usage of self-aggregation algorithms to support load balancing in a highly dynamic and pervasive setting. A preliminary description

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Bionetics '08, November 25-28, 2008, Hyogo, Japan
Copyright 2008 ICST 978-963-9799-35-6.

of our work is reported in poster [14]. Here we describe the approach in more detail and evaluate it through simulation. In particular, our simulation experiments show how the application of self-aggregation algorithms enables load balancing also in the presence of dynamic networks, heterogeneous types, and variable workload and processing time. Besides, our experiments show that the introduction of self-aggregation does not introduce a significant overhead in terms of execution time, even if it requires the exchange of a higher number of messages between nodes.

The organization of the paper is as follows. Section 2 presents some existing dynamic load balancing algorithms. Section 3 describes the aggregation problem and presents some distributed algorithms that address it. In Section 4 we discuss on how to apply self-aggregation for load balancing. Section 5 presents the results we have obtained by simulating the proposed approach. Section 6 presents an overview of the state of the art. Finally, Section 7 concludes the paper.

2. DYNAMIC LOAD BALANCING

In this section we introduce the load-balancing problem in the context of decentralized reconfigurable distributed systems. In particular we focus on some existing dynamic load-balancing algorithms and how it is possible to accelerate their converge in overlay networks with multiple types of nodes.

Load balancing algorithms can be broadly classified into two categories, static and dynamic [23]. Here we focus on the dynamic approaches that defer the decision on how to distribute the jobs at runtime, based on dynamic information instead of static one.

Assume a network of interconnected nodes. Each node can be seen as a resource that is able to process jobs. Each node corresponds to a type that defines which job(s) it is able to process. In this kind of networks the purpose of Load-Balancing is to distribute the jobs evenly to all the nodes of the network with the aim to:

- increase the job processing rate of the whole network;
- increase the number of nodes involved in a computation, and reduce, at the same time, their utilization.

Extensive studies have been carried out on this topic applied to networks of interconnected microprocessors and generic parallel computing architectures. Some studies focused on centralized architectures in which there is an entity that decides the assignment of the jobs, others on decentralized architectures where each node takes the decision to pass or enqueue a job using its available information. In decentralized architectures a node may have the following type of knowledge:

- *"a priori" static knowledge* of network topology, such as the probability of jobs distribution, static route tables, etc;
- *local dynamic knowledge* of the list of neighbors, including their type and their workload;
- *global dynamic knowledge* of the global status of the network through a shared memory environment.

In the context of this paper we consider only the decentralized architectures since we want a solution without single points of failure that scales well with minimal configuration issues when increasing the network size. Another requirement is that a node should not have global and *a priori* knowledge of the network: this means that the decisions on how to assign jobs would be taken on the basis of local information only.

In the literature there are two algorithms that, using simple local rules and knowledge, are able to balance the workload of a network. They are the Diffusive Load Balancing Algorithm and the Dimension Exchange algorithm [10]. Both of them have been formally studied and their convergence has been mathematically proved in [25]. Variations of these exist as well (see for instance [30]), but they speed up the load balancing convergence only in presence of some particular invariant properties, such as having a fixed network topology or other predictable patterns. Thus, we do not consider them since we do not want to make any assumption on the evolution of the network that can arbitrarily evolve gaining and losing links and nodes.

The Diffusive Load Balancing Algorithm is described by the following formula that determines the workload w that node i should have at time $t+1$ considering its workload at time t as well as the workload of its neighbors at the same time t:

$$w_i(t+1) = w_i(t) + \sum_j \alpha_{ij}(w_j(t) - w_i(t))$$

In the formula, i is a generic node of the network, $1 \leq j \leq degree(i)$ identifies each neighbor of i[1], and α_{ij} is an algorithm parameter that is usually equal to the inverse of the degree of node i incremented by one. Based on this formula it results that the load of all the nodes in the network tends to be evenly distributed after some time interval. Figure 1 shows an example of how a network of four nodes would evolve starting from the situation in which one of the nodes owns all the load of the network. In the figure each circle represents a node, each star represents a job. After the network has been initialized, a node randomly activates (black circle), and locks its neighbors (gray circles). Then, it retrieves information about the number of jobs of its neighbors, and, based on that, moves the jobs so to equilibrate the load within its neighbors. Finally, the active node unlocks the others. Iterations are repeated and can occur also simultaneously until a steady, load balanced, state is reached. The main limit of this approach is that each activated node requires to interact with all its neighbors in the same iteration.

The Dimension Exchange Algorithm is described by the following formula:

$$w_i(t+1) = w_i(t) + \frac{w_j(t) - w_i(t)}{2}$$

where i is a generic node and j is a random neighbor of node i. At the end of the iteration, both nodes will have the same number of jobs. Figure 2 shows an example of application of such algorithm. Its advantages, compared to the Diffusive Load Balancing Algorithm, reside in the fact that at any iteration the interaction is limited only to pairs of nodes. This makes the algorithm less sensitive to synchronization issues and more suitable in the case of dynamically reconfigurable networks where the node degree changes significantly over time. For the above reasons in our work we

[1] $degree(i)$ defines the number of neighbors of i.

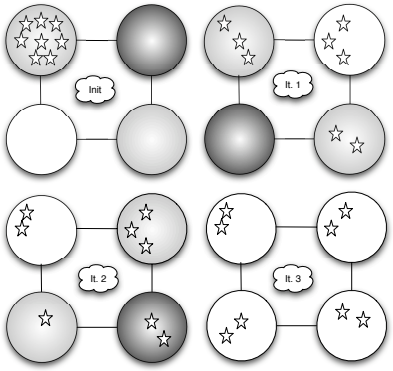


Figure 1: Example of execution of Diffusive Load Balancing Algorithm.

have decided to exploit this algorithm over the other. The

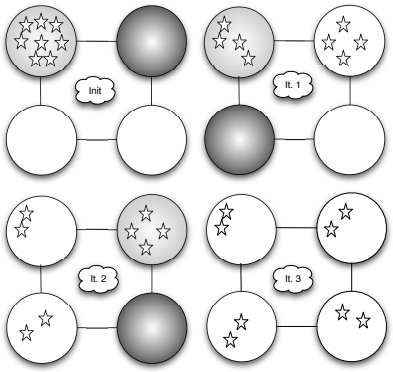


Figure 2: Example of execution of Dimension Exchange Load Balancing Algorithm.

presented algorithms consider a network of interconnected nodes in which the topology can dynamically change but they do not conceive the possibility to have various nodes and jobs of different types coexisting in the same network (*heterogeneous case*). In this case, if, for example, a node of type A has 10 jobs of type A to balance, but its neighbors are all (or almost all) of type B, the load balancing algorithm is not able to work properly even in the case other nodes of type A exist in the network. This is due to the implicit assumption that this kind of algorithms work only on homogeneous network domains, where a homogeneous domain is defined as a connected subgraph of the original network that is composed only nodes of a single type.

We devise the following strategies to solve the load balancing problem in heterogeneous networks:

- make the jobs traverse the incompatible nodes;
- modify the links of the network (rewire) in order to aggregate the nodes of the same type to form a single domain.

The first solution is not applicable because the nodes are not able to forward the jobs directly to their target since they do not have enough global information about the network. With the usage of a random policy it is possible that all the jobs are eventually processed, but there is always a possibility that the convergence in extreme situations is worse than not using load-balancing at all.

In the second method we need to identify a proper algorithm to achieve such node aggregation without global knowledge on the structure of the network. The consequence is that, after the aggregation process, the Dimension Exchange Algorithm will behave in the case with heterogeneous node types as efficiently as in the case with homogeneous node types.

In the following section we describe a way in which each node is able to modify the connections with its neighbors in order to reach a more efficient and effective configuration, while in Section 4 we describe the application of this approach to the balancing of workload.

3. SELF-AGGREGATION ALGORITHMS

The final purpose of the self-aggregation algorithms we are presenting is to rewire the network with the aim to reduce the number of links from *incompatible* nodes and to add new links to *compatible* nodes. The notion of compatibility is related to the type of the nodes. In our case we refer to a specific case of self-aggregation that is named *Clustering*. In this case nodes tend to establish links with neighbors of the same type. Therefore, compatibility between two nodes is defined as the equality of their types.

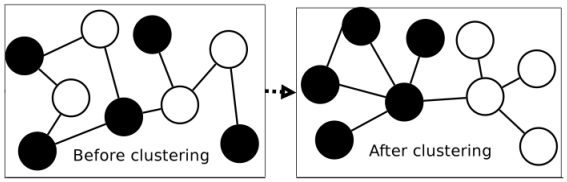


Figure 3: The effect of clustering on a set of nodes. Colors indicate the type of nodes.

Figure 3 shows the situation of a network of two types of nodes (depicted by the back and white colors) before and after the execution of the clustering algorithm. As it can be seen from the picture, the network tends to organize in two areas, one containing nodes of type black and the other nodes of type white. These algorithms require to run continuously in order to maintain a dynamic system in the desired state even in unpredictable situations like independent arrival and departure of nodes (this is called *churn* in peer-to-peer networks [28]) due to their dynamic nature. In contrast to what happens with other approaches like the one in [29] the network may possibly be partitioned, but such partitioning will not be permanent since the continuous appearance of new nodes will keep adding random links to the network.

While clustering algorithms, per se, are not new in the literature (see the state of the art analysis presented in Section 6), here the interesting aspect is that clustering is not executed by a centralized entity, external to the network. Instead, it is executed in a distributed way thanks to the ability of each node to autonomously take a simple "disconnect/maintain the link" decision on the basis of the type of each neighbor.

In [13] and [24] we have presented the self-aggregation algorithms we have identified so far. Here we briefly summarize them and in the next sections we show how they could be used to support load balancing.

The first algorithm we have studied is named Active Clustering [24]. It is based on the iterative execution by each node in the network of the following steps:

- at a random time the node elects itself as the *initiator* node and elects a *matchmaker* node among its neighbors;
- the matchmaker node chooses one neighbor that is compatible with the initiator and makes the two establish a new link;
- finally, the matchmaker removes a link between itself and the chosen neighbor.

The performance of this algorithm, in the following called *Original Saffre Clustering*, has been evaluated in [24]. The results clearly show that the system tends to reach a steady state. Starting from these results we have tried to understand why such simple and local laws are able to organize complex networks with the aim of investigating possible further optimizations. The most interesting fact we have noticed is that the previous algorithm does not always perform operations that increase the number of links between compatible nodes. In this case it is said that the algorithm introduces some *noise* into the system. In [21] for example, it is explained that in the biological world this noise is necessary because, on large numbers, this increases the probability to obtain an optimal solution. To understand if this observation holds in our context, in [13] we have investigated the effects of an increase or a decrease of noise in the original self-aggregation algorithms.

We have first removed all the noise from the original algorithms: this resulted in a new algorithm that we call *Fast Algorithm*. It is similar to the original one, but with the additional constraint that an algorithm iteration can never remove a link between compatible nodes. From the preliminary simulations we have seen that, with respect to the original algorithm, this one has a faster convergence rate because it avoids "noisy" iterations. Also, another advantage is that it reduces the total number of link exchanges because of the lower number of neighbors that can be chosen. The disadvantage is that the increase in the number of links between compatible nodes is not as good as in the original clustering. This leads us to conclude that the noise is a key factor for the accuracy of the algorithm.

The second investigation we have done has been to increase the algorithm noise. This case, that we have called *Accurate*, assumes that the decision of adding and removing links in each algorithm iteration is fully unconstrained, except for the fact that the total number of links must remain the same and that a link between incompatible nodes can be added only if a link between incompatible nodes is removed in the same iteration. This constraint ensures that the aggregation level of the system in the worst case remains constant and never decreases. After the preliminary simulations we have seen a lower convergence rate and a larger number of exchanged messages with respect to the original algorithm. However the number of links between compatible nodes has increased. This strategy is similar to what happens in genetic algorithms [16]: in a genetic algorithm each iteration has a mutation operation that randomly modifies the solutions that are computed until that moment. This prevents the genetic algorithm to get stuck in local optima and therefore improves the accuracy.

Given the advantages and disadvantages offered by each solution, we have defined a *self-adaptive* algorithm that is able to modify its behavior according to some local rules. These local rules have been modeled as a Finite State Machine (FSM). As shown in Figure 4, the general logic is that the algorithm starts behaving as the most constrained algorithm (Fast Algorithm) and stays in that state until the constraints inhibit further iterations (this happens when a node gets stuck because it does not have neighbors to choose that satisfy the algorithm requirements). In such a case the algorithm switches to a medium-constrained algorithm (Original Saffre Clustering) first and then, if it gets stuck again, to the less constrained algorithm (Accurate Clustering). Finally, as soon as a new neighbor is added in a local node, it switches again to the most constrained algorithm.

In Figure 4 *Failure* transition is triggered when an algorithm is not able to complete an iteration because of its constraints. *Success* transition is triggered when an iteration terminates successfully. *New Neighbors* transition is triggered when a new neighbor has been added to the local node.

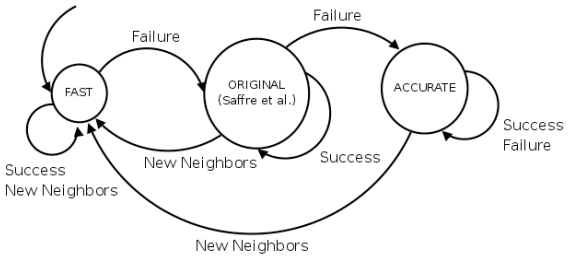


Figure 4: Adaptive Clustering Algorithm FSM

4. SELF-AGGREGATION FOR LB

As discussed in Section 2, to overcome the inherent limitations of classical load balancing algorithms, we argue that autonomic self-aggregation techniques could help since they rewire the system in groups of homogeneous nodes that are then able to balance the load among each others using classical techniques. The algorithm we have chosen for the reconfiguration of network topology is the Adaptive Clustering Algorithm presented in Section 3. This algorithm runs in parallel with the Load Balancing algorithm in order to enhance its convergence rate, and therefore maximize the throughput of the system. More precisely, it is started when the network is created and stays active forever. The only information it uses and modifies is the list of neighbors of each node involved in an iteration. In parallel, the Dimension Exchange Algorithm is activated when a node has in its neighbors list at least a node of the same type and its queue of jobs is not empty. It can modify only its list of queued jobs and the one of its neighbors.

Since the two algorithms modify always different node proprieties, no conflicts are possible, therefore they can be executed in parallel without the need of coordinating them. An example of execution of both algorithms can be seen in Figure 5. To make the example understandable to the reader, the clustering and load balancing algorithms are shown as interleaved, but in reality they actually are executed in par-

allel. In the figure circles represent nodes, circle color represents the type of the node, and the number reported inside represents the current number of jobs contained in the node. The nodes that are activated by the iterations of one of the algorithms are depicted with a double border. In the example we can see that, after applying load-balancing to the initial configuration, we reach a steady state in which the load is balanced at the local level, but not at global level.

After a clustering iteration, some previously separate domains are possibly joined, therefore another load balancing iteration is able to further improve the balance of the load. If we iterate this process we reach the last configuration of Figure 5 in which the network is fully clustered and the workload is balanced at global level.

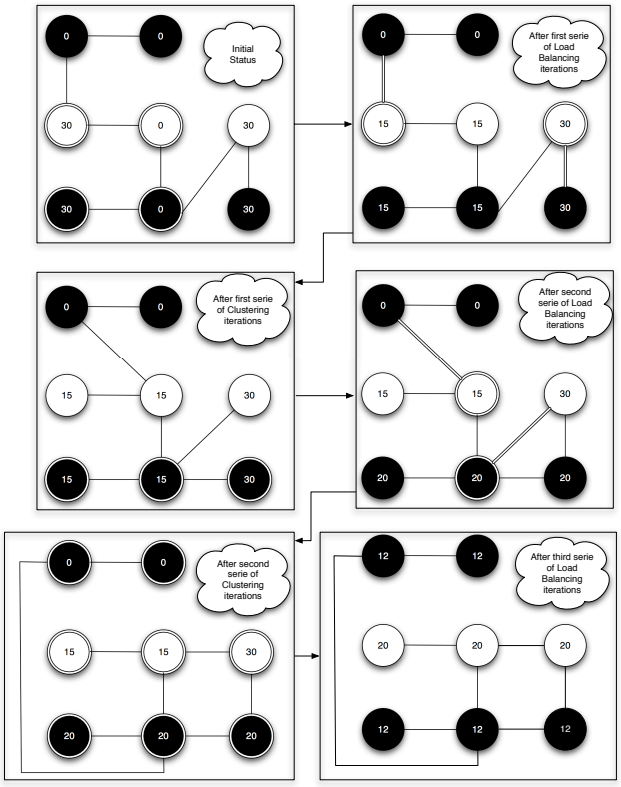


Figure 5: Simultaneous execution of Active Adaptive Clustering with the Dimension Exchange Load Balancing algorithm.

Of course, during the execution of the algorithms the network can evolve: new nodes can appear and connect to some already existing nodes and others can disappear. Since self-aggregation is always running, this do not cause severe perturbations to the whole system as we show in Section 5 , provided that the network continues to stay connected.

5. EXPERIMENTAL RESULTS

In this section we show the experimental results we have obtained using a combination of load balancing and the previously introduced self-aggregation techniques on dynamic and distributed networks with different types of nodes.

We first describe the simulation environment and the performance parameters that have been used to quantify the level of load balancing of the network (Section 5.1). Then we describe the results we have obtained in Section 5.2. Finally, we discuss about the strengths and the weaknesses of this approach in terms of trade-off between convergence speed and the message overhead that is added by the rewiring operations (Section 5.3).

5.1 Setting up the Experiments

Simulation Environment. To set up the experiments we have used a simulation framework that we have implemented for this specific purpose. All the results that will be presented in this section have been produced by Monte Carlo simulations that have been repeated at least 20 times, to give statistical significance to the obtained data. In our previous study presented in [13] we have shown that the adaptive clustering algorithm is scalable with respect to the number of nodes and links, therefore in the simulations we will start with an initial number of nodes equal to 100 and an average node degree equal to 4. Other values might be chosen without affecting the results in a considerable way. The initial topology that has been used is the Scale-Free one [4] since it is the most similar to a real network. The heterogeneity of the network has been fixed to 10% to have a reasonably difficult scenario for the load balancing problem, therefore we have 10 different types of nodes and jobs, with the constraint that a job cannot be assigned or processed by a node that does not match its type.

The load of the network and the node processing time vary over time. Indeed, the network structure changes dynamically in terms of nodes and links among them. Thus, the following input parameters have been defined:

- *Jobs distribution*: this parameter indicates how the jobs are distributed among the nodes. We have considered the case with an initial static workload of 400 jobs and the case of a run-time insertion of other 400 jobs every 20 seconds. All the jobs enter the system in the most unbalanced way: they are sent to a single randomly selected node for each type.

- *Node processing time*: this parameter specifies the job execution time for the nodes of the network. We have considered the situation in which all the nodes are able to process the jobs in 5 seconds, and the case in which 70% of the nodes require 7 seconds and 30% 3 seconds; in both situation the maximum ideal throughput is equal to 20 jobs/second.

- *Node churn*: this parameter represents the rate in which nodes can appear and disappear in the network. We will show the situation without node churn and the situation in which every 10 seconds 20% of the nodes are removed from the network and replaced with other nodes with new random links. The new nodes are created in such a way that the cardinality of nodes, links, types, and jobs tends to stay similar to the original one.

Performance Parameters. To study the performance of the load balancing algorithms the following three performance parameters have been considered:

- the overall *Number of completed jobs*,

- $Throughput = CompletedJobs/ElapsedTime$, and

- the *Average Number of messages exchanged by each node* that counts the number of messages that are exchanged by each node since the beginning of the simulation.

For each performance parameter we have considered its minimum, its average, and its maximum.

5.2 Results

In Figure 6 we can see the most important results of our experiments.

Load-Balancing without Rewiring. The experiment in Figure 6a has been done to show how the basic distributed load balancing algorithm behaves in a network with different types. In this case rewiring is not executed. The obtained results are similar to what we would ideally have in a fully disconnected network: 10 jobs processed every 5 seconds. This result is not unexpected since the load balancing approach we have used has been designed to work in homogeneous networks only, in fact the actual load-balancing happens only in small domains of the network as we predicted in Section 4. In other words further load balancing iterations are inhibited by the fact that the jobs cannot traverse the nodes with different types.

Load-Balancing with Rewiring. In the experiment of Figure 6b we used the same parameters of the previous one with the addition of the parallel execution of the Active Adaptive Clustering rewiring algorithm on each node. The results show a significant improvement with respect to the one in Figure 6a, although it is slightly worse than the ideal one. Looking at the minimum and maximum number of processed jobs we can see that there is a slight difference between them and the average number of processed jobs in the middle steps of the simulations, while this difference is irrelevant in the first steps and the last steps of the simulation (after all initial jobs have been processed). This phenomenon can be explained by the fact that every simulation is started with a different topology seed, and each node evolves in a different way even if it executes always the same simple rules.

Load-Balancing with Rewiring and Multiple Bursts. In this last series of experiments we study how the network evolves when it keeps on receiving bursts of new jobs on different nodes at fixed time intervals. This makes it possible to point out how the combination of load-balancing and clustering can significantly improve the throughput of the system, leading to values that are more typical of the ones ideally obtained in centralized optimal load-balanced systems. The results of these simulations are shown in Figure 6c in terms of number of processed jobs over time. The values in the initial simulation steps are similar to the ones in Figure 6b. Then, the curve continues to grow as a straight line because of the continuous job bursts that are sent to the nodes. The measured throughput is shown in Figure 6d. It is close to the optimal value because, as long as new jobs arrive, the rewiring algorithm modifies the network in a way that leads to the creation and enlargement of "island of nodes" of the same types. Once created, these islands are then able to apply the load-balancing techniques like in a homogeneous (single type) configuration, that is the ideal scenario for this kind of algorithms. Figures 6e and 6f show respectively a situation in which we evaluate the throughput of the algorithm in presence of a node churn (arrival/departure) of 10% every 10 seconds, and when the nodes have different job processing power (30% of the nodes process a job in 3s, 70% of the nodes in 7s) but the same overall ideal throughput. In both experiments we obtain throughputs similar to the one in Figure 6d. This means that the algorithm is able to reconfigure the network in such a way that it becomes easier for a node to find a compatible node for sharing its workload.

Network overhead. When we use rewiring the number of messages exchanged by nodes increase compared to the plain application of load balancing because of the need to build and preserve the clustered topology. In all our experiments the rewiring communication overhead was independent from the evolution of the network and was constantly equal to 5 rewiring messages per second per node. Instead, the number of messages needed by the load balancing algorithm was much smaller, equal to 0.033 messages/second. However, the lower number of messages could be compensated by the size of each message. In the case of rewiring such size is constant and very small, while in the case of load balancing it can depend on the size of jobs if these need to be transfered from node to node.

5.3 Discussion

As we have argued in the previous section, the adoption of rewiring introduces some network overhead. Such overhead can be kept under control by reducing the frequency of rewiring iterations, of course, at the expenses of the structure of the various network domains. In general, properly dimensioning such frequency is a design problem that depends on the nature of the specific application domain and especially on the network heterogeneity and node churn. This study, and particularly the penalties associated with rewiring, will be the subject of future work.

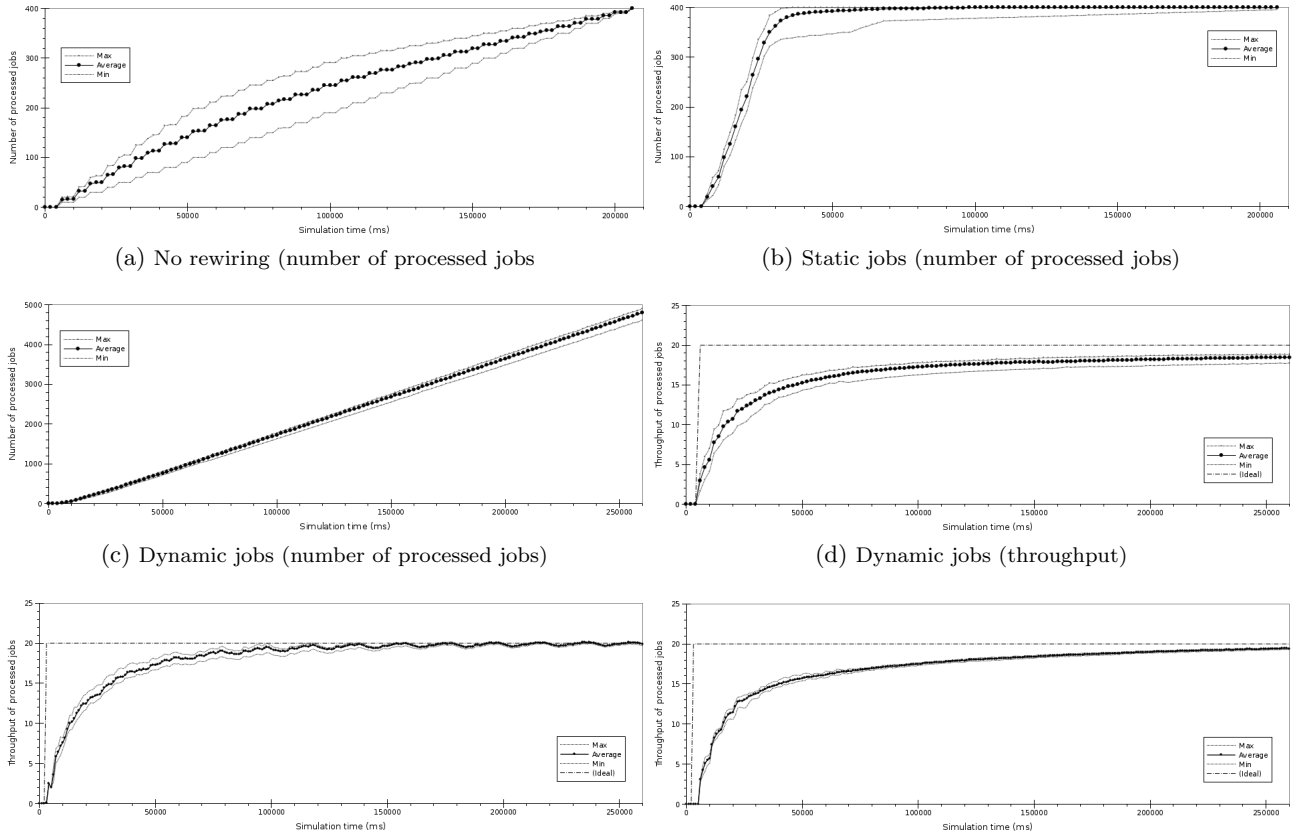
From the simulation results we can see that the proposed approach shows self-healing proprieties with respect to the organization of its topology. This is especially true in presence of churn, where the constant self-reconfiguration allows the network to continue balancing and executing its jobs. It is particularly interesting to notice that having such appearing/disappearing of nodes introduces some additional randomness into the system that is also able to improve slightly its performance since our churn experiments kept the number of nodes, jobs, and links constant. We expect that an increase or a decrease in the number of links/nodes would impact the results in a proportional way, this will be the subject of a future study.

The most critical limitation of our approach is the fact that in the case of particular topologies the churn may remove some of the nodes with the highest degree. In this situation the network will be split into isolated domains that cannot exchange their jobs. However the dynamicity of the network, especially the arrival of new nodes, can make this problem transitory.

6. RELATED WORK

In this section we briefly review the current approaches in the areas covered by our approach, namely self-aggregation and load balancing techniques.

Self-aggregation techniques are based on the principle of identifying homogeneous subgroups of cases in a population. The responsibility among the individual entities is the following: no single entity is in charge of the overall aggregation, but each contributes to a collective behavior. Following this philosophy, mainly inherited from natural adaptive sys-



(a) No rewiring (number of processed jobs

(b) Static jobs (number of processed jobs)

(c) Dynamic jobs (number of processed jobs)

(d) Dynamic jobs (throughput)

(e) Dynamic jobs with churn (throughput)

(f) Dynamic jobs with different service time (throughput)

Figure 6: Experiments. (a) shows the number of processed jobs in experiments that have been performed without rewiring; (b) shows an experiment with rewiring and a fixed static load of jobs; (c), (d) show what happens to the situation in (b) if we keep adding jobs to the nodes; (e) and (f) show how the throughput is affected by network churn or by different processing times among the nodes.

tems, the local behavior rules applied in all entities lead, with a certain probability, to the desired global behavior. Examples of application of these rules can be found in the area of communication networks: for example for the control of the topology in wireless multi-hop networks [6], or the computation of a maximal independent set in radio networks [18]. Several kinds of applications of self-organization techniques in communication networks can be found in [9, 22, 19]. Moreover, in the research area of peer-to-peer computing there are different approaches like Cyclon [29] protocol in which self-organization of the overlay is achieved by shuffling nodes' neighbors periodically, thus achieving a connected overlay without disrupting randomness.

Different lines of research apply methods based on the use of genetic algorithms or neural networks to define and to study the problems related to cluster formation, e.g., [1] or the multi-agent approach like in [26].

Load Balancing The literature in the load balancing domain has been focusing mainly on networks composed of homogeneous nodes. In this context, [3] is investigating dynamic networks by extending the classical load balancing algorithm to this purpose. Different approaches have been proposed in [20, 11] in which load balancing is achieved in a wireless ad hoc network using the routing infrastructure, rather than changing the logical overlay as we have proposed in this work. Indeed, also in these cases nodes are assumed to be homogeneous. Nodes are still homogeneous in [17, 8] that propose bio-inspired load balancing algorithms.

[15] deals with heterogeneity interpreted as the different processing power of nodes. Instead, we say that nodes are heterogeneous if they can process different types of jobs.

7. CONCLUSIONS

In this paper we have introduced the usage of self-aggregation algorithms to support load balancing in the context of highly dynamic and distributed systems. The use of these algorithms makes it possible to dynamically create and maintain groups of similar nodes that are able to know their neighbors and to execute the needed load balancing algorithms.

The study of these combined algorithms has been performed by simulating their execution through a distributed simulator.

Our experiments show that the introduction of self-aggregation improves the overall load balancing and does not introduce a significant overhead in terms of execution time, even if it requires the exchange of a higher number of messages between nodes.

The work can be expanded in several directions, serving as a base upon which more refined architectures and im-

plementations can be developed. More adaptive clustering algorithms can be defined taking into account changing environments and goals. The combination of load balancing and clustering algorithms can also be generalized so to be able to deal with domains in which the nodes have multiple types. Another future work would be to support these algorithm with a mechanism to dynamically adapt the frequency of the algorithm iterations, keeping into account the cost of sending messages, the convergence time, and other quality parameters.

Acknowledgments

This work has been partially supported by Project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

8. REFERENCES

[1] H. B. Amor and A. Rettinger. Intelligent exploration for genetic algorithms: using self-organizing maps in evolutionary computation. In *GECCO 2005*. ACM, 2005.

[2] O. Babaoglu, H. Meling, and A. Montressor. Anthill: A framework for the development of agent-based peer-to-peer systems. *ICDCS*, 2002.

[3] J. M. Bahi, R. Couturier, and F. Vernier. Synchronous distributed load balancing on totally dynamic networks. In *IPDPS07, Long Beach, California, USA*, pages 1–8, 2007.

[4] A.-L. Barabasi and A. Reka. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[5] R. Beckers, O. Holland, and J. Deneubourg. From local actions to global tasks: stigmergy and collective robotics. In *ALIFE IV*. Brooks & P. Maes, MIT Press, Cambridge (Mass), 1994.

[6] D. M. Blough and al. The lit k-neigh protocol for symmetric topology control in ad hoc networks. In *MobiHoc*. ACM, 2003.

[7] S. Camazine and al. Self-organization in biological systems. *Princeton University Press, Princeton*, 2001.

[8] G. Canright, A. Deutsch, and T. Urnes. Chemotaxis-inspired load balancing. In *European Conference on Complex Systems*, Nov. 2005.

[9] G. D. Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *JAIR*, 9:317–365, 1998.

[10] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7(2):279–301, 1989.

[11] B. Danila, Y. Yu, S. Earl, J. Marsh, Z. Toroczkai, and K. Bassler. Congestion-gradient driven transport on complex networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 74(4):046114, Oct. 2006.

[12] C. Detrain, J. Deneubourg, and J. Pasteels. Information processing in social insects. *Birkhauser, Basel*, 1999.

[13] E. Di Nitto, D. J. Dubois, and R. Mirandola. Self-Aggregation Algorithms for Autonomic Systems. In *Bionetics '07*, Budapest, Hungary, December 2007.

[14] E. Di Nitto, D. J. Dubois, R. Mirandola, F. Saffre, and R. Tateson. Self-Aggregation Techniques for Load Balancing in Distributed Systems. In *SASO '08*, Venice, Italy, October 2008.

[15] R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35(3):305–320, 2002.

[16] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[17] M. Jelasity, A. Montresor, and Ö. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In *Engineering Self-Organising Systems*, volume 2977 of *LNCS*, pages 265–282. Springer, 2004.

[18] T. Moscibroda and R. Wattenhofer. Efficient computation of maximal independ sets in unstructured multi-hop radio networks. In *Proceedings of IEEE International conference on Mobile Ad-hoc and Sensors systems*. IEEE, 2004.

[19] T. Nakano and T. Suda. Applying biological principles to designs of network services. *Appl. Soft Comput.*, 7(3):870–878, 2007.

[20] N. Nehra, R. B. Patel, and V. K. Bhat. Routing with load balancing in ad hoc network: A mobile agent approach. In *ACIS-ICIS*, pages 489–495. IEEE Computer Society, 2007.

[21] J. Pasteels, J. Deneubourg, and S. Goss. Self-organization mechanisms in ant societies (i) : trail recruitment to newly discovered food sources. In *From individual to collective behavior in social insects*, pages 54, 155–175, BirkhaÃijser, Basel, 1987. Eds J.M. Pasteels & J.L. Deneubourg. Experientia Supplementum.

[22] C. Prenhofer and C. Bettstetter. Self-organization in communication networks:principles and design paradigms. *IEEE communication magazine*, 2005.

[23] X. Qian and Q. Yang. An analytical model for load balancing on symmetric multiprocessor systems. *J. Parallel Distrib. Comput.*, 20(2):198–211, 1994.

[24] F. Saffre, R. Tateson, J. Halloy, M. Shackleton, and J. L. Deneubourg. Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications. *The Computer Journal*, 2008.

[25] P. Sanders. Analysis of nearest neighbor load balancing algorithms for random loads. *Parallel Comput.*, 25(8):1013–1033, 1999.

[26] G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, editors. *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering*, volume 2977 of *LNCS*. Springer, 2004.

[27] M. Shackleton and P. Marrow. Editorial, special issue on nature-inspired computation. *BT Technology Journal*, 18(4):9–11, October 2000.

[28] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *IMC*, pages 189–202, New York, NY, USA, 2006. ACM.

[29] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2), 2005.

[30] C.-Z. Z. Xu and F. C. M. Lau. The generalized dimension exchange method for load balancing in κ-ary n-cubes and variants. *Journal of Parallel and Distributed Computing*, 24(1):72–85, 1995.