A Bio-Inspired Algorithm for Energy Optimization in a Self-organizing Data Center

Donato Barbagallo, Elisabetta Di Nitto,
Daniel J. Dubois, and Raffaela Mirandola

Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32, 20133, Milano, Italy
{barbagallo,dinitto,dubois,mirandola}@elet.polimi.it

Abstract. The dimension of modern distributed systems is growing everyday. This phenomenon has generated several management problems due to the increase in complexity and in the needs of energy. Self-organizing architectures showed to be able to deal with this complexity by making global system features emerge without central control or the need of excessive computational power. Up to now research has been mainly focusing on identifying self-* techniques that operate during the achievement of the regular functional goals of software. Little effort, however, has been put on finding effective methods for energy usage optimization. Our work focuses specifically on this aspect and proposes a bio-inspired self-organization algorithm to redistribute load among servers in data centers. We show, first, how the algorithm redistributes the load, thus allowing a better energy management by turning off servers, and, second, how it may be integrated in a self-organizing architecture. The approach naturally complements existing self-management capabilities of a distributed self-organizing architecture, and provides a solution that is able to work even for very large systems.

Keywords: energy optimization, bio-inspired algorithms, autonomic computing, self-organization, data centers.

1 Introduction

The constant growth of energy usage in industrialized countries is creating problems to the sustainability of the Earth development. The problem of energy use concerns many fields in human activities, for this reason some new disciplines such as green computing are growing up to study how to consume less energy by providing the same quality of service.

In general, research in energy saving focuses towards two different directions:

- developing new technologies that need less energy to work, e.g. energy saving lamps;
- developing new techniques to make a rational and efficient use of existing tools, e.g., car sharing is a way to use cars in a more sustainable way.

In the context of software engineering, the first direction is addressed by the development of new software architectures and algorithms that are able to support energy saving, while the second one corresponds to new management processes that may be used to instrument existing architectures and algorithms to achieve a final energy reduction without having to modify or replace existing software components.

In this work we present two distinct contributions: the first is an algorithm for reducing power consumption in a large data center, the second is the integration of this algorithm into a decentralized platform supporting the management of the applications installed in the data center.

The general idea for the algorithm is to move the load from a server to another one to maximize the power efficiency of the whole data center. The power efficiency is defined as the available computational power over the energetic power required to provide such computational power. In other words the goal of the algorithm is to bring the network from a situation of randomly distributed load to a situation in which part of the servers are used with the maximum efficiency, while other servers are turned off to have a maximum reduction of wasted power. The algorithm we have developed takes some ideas from the biological world such as the explorative behavior of swarming insects in their search for a better place [1]: in our context migrating entities are the virtual machines of each server (for us they represent the server load), while the possible migration sites are the servers.

For what concerns the architectural integration problem, the approach we propose starts from the use of an existing autonomic framework (SelfLet architecture [2, 3]). The reason for this choice is that such architecture is highly decentralized and therefore scalable to the dimensions of a large data center.

In order to assess our approach, we have set up some preliminary experiments that show the algorithm behavior under different conditions. Based on the results, we can argue that our approach inherits some classical benefits from bio-inspired solutions such as scalability, complete decentralization, and capability to achieve good results even in extreme conditions.

The remainder of this paper is structured as follows: Section 2 shows the motivations and the objectives of this work; Section 3 describes the technologies we are using, and in particular it introduces the architectural model and the algorithm model; Section 4 depicts the integration work that has been done both on the architectural and on the algorithmic level; Section 5 reports and discusses the simulations that have been carried out to test our approach. Section 6 presents the most representative state-of-the-art works aimed at reducing power consumptions in data centers. Finally, Section 7 depicts conclusions and future works.

2 Motivations and Approach

In the past the focus of microelectronics industry has been the miniaturization of components and the design of better performing devices. Such an evolution

led to very fast IT systems. However, these systems are often used inefficiently [4].

As shown in [5], the interest towards efficient use of technology is motivated by some alarming trends:

- recent researches have demonstrated how IT is responsible for more than 2% of global CO_2 emissions [6–8];
- power is the second-highest operating cost in 70% of all data centers;
- data centers are responsible for the emission of tens of millions of metric tons of carbon dioxide annually, more than 5% of the total global emissions.

Figure 1 shows the partition of global expenditure for servers from 1996 to 2010 as predicted by Josselyin *et al.* [9]. From an economical perspective it can be noticed that whereas the cost of hardware has only slightly grown in the last 12 years, the cost of power and cooling has grown four times.

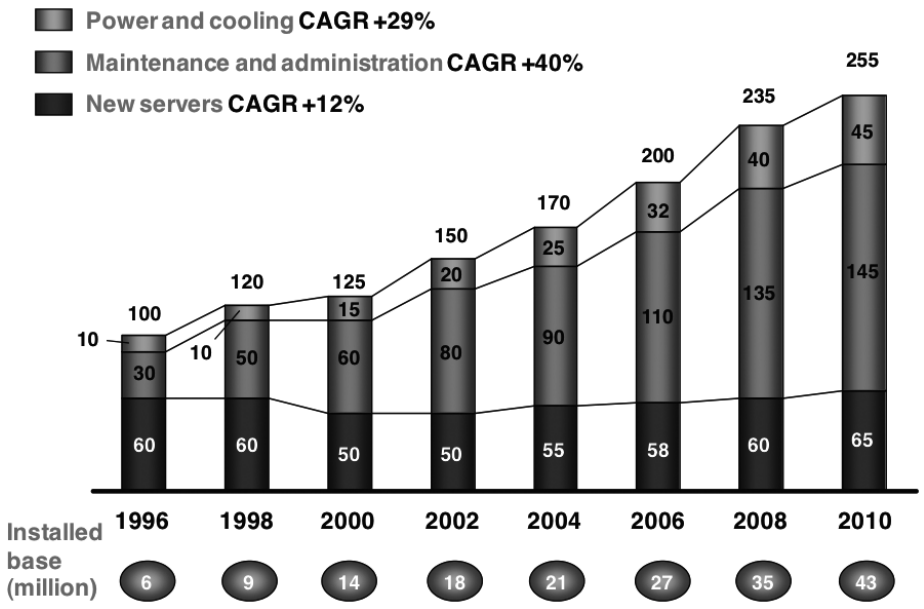


Fig. 1. Global spending for servers, prediction by [9].

As described in [10], data centers are found in nearly every sector of the economy, including financial services, media, high-tech, universities, and government institutions. Dramatic servers growth in data centers is indicated by well-known web services such as Google, Amazon, and eBay. Estimates indicate that Google maintains more than 450,000 servers, arranged in several data centers located around the world. According to the research firm IDC [9] by 2012, for every 1$

spent on hardware, 1$ will be spent on power and cooling. For this reason, much of the interest in Green IT comes from the financial return on green data center investments.

In order to follow these new needs of companies, also hardware manufacturers are studying more efficient solutions. In 2007 some big companies like AMD, Dell, IBM, Sun Microsystems, and VMware have founded The Green Grid, a global consortium which has the goal to investigate new green-oriented solutions [11].

The work in [10] indicates several dimensions, which should be exploited when trying to obtain good green results. Such dimensions are divided into organizational tasks (e.g., the creation of a responsible position for the Green IT), technical tasks (e.g., measuring the power-usage effectiveness), and technological tasks (e.g., consolidation and virtualization). The last mentioned one is the one that motivates the approach we are proposing. Virtualization allows a high decoupling between the application layer and the physical layer of any existing system. This allows applications to be moved and executed on different types of hardware, thus enabling the possibility of optimizing power consumption without the need to modify the applications themselves. The next two paragraphs describe more in detail the motivations of the architectural and algorithmic choices we have made in our work.

2.1 The Importance of Self-organizing Architectures

The solution we propose is framed into the Autonomic Computing field. The term Autonomic applied to the IT sector has been first used by IBM [12] to mean systems able to self-manage, self-configure, self-protect, and self-repair without any user intervention. Most architectures that follow this paradigm tend to add a supervision layer to existing software architectures that is able to "reason" using the data provided by the underlying system, and to "react" by ordering the system to perform some sort of actions. This last approach has the problem of keeping all the complexity concentrated in this supervising layer, therefore more recent architectures tend to offer the opportunity to distribute such complexity to all the elements of the system. In such a way the system is able to achieve complex global behaviors using only simple rules at the level of each component. This last approach is the one at the basis of our work since we are dealing with systems that may have thousands of nodes and where system self-reorganization should occur spontaneously. In particular our reference architecture is the SelfLet one that will be described in detail in Section 3.

2.2 The Importance of Self-organizing Algorithms

The architectures described above provide software primitives for interfacing with existing systems, as well for monitoring events, taking actions, and exchanging messages. However they are completely useless without a self-organizational logic that defines the goals to achieve and algorithms to achieve these goals.

For the purpose of this work we have used a bio-inspired algorithm since this class of algorithms is usually intrinsically self-organizing and characterized by

scalable and fault-tolerant behavior. This kind of algorithms is particularly important because their final results often emerge from individual behaviors of the "colony" that is running the algorithm (the most common example is the Ant Colony Optimization algorithm [13]). The main drawback of these algorithms is that they are usually expressed in such a way that even if they can be modeled and studied in a theoretical way, there are many assumptions on the running architecture that make difficult to integrate them into existing self-organizing systems. Moreover there are too few attempts of using these techniques in research targeted on Green IT.

The idea behind this work is to model the data center as a peer-to-peer network of nodes that collaborate to each other using a bio-inspired algorithm based on the scout-worker migration method described in [14]. In this algorithm some entities of the system are workers (which correspond to the virtual machines) and other entities are scouts (which are entities allowed to move from one physical node to another to collect information): the information collected by the scouts is then used by workers to take corrective actions (such as the migration of workers, and the switching-on and off of physical machines). This algorithm is particularly suitable in decentralized systems scenarios for the following reason: the absence of global coordinators, the absence of global knowledge, and the fact that choices are emergent perfectly fit the situation in which the data center is characterized by large dimensions (up to thousands servers, like in the case of telecommunication companies or governmental agencies) and high dynamism (e.g., requests peaks or machine failures). A more detailed description of this algorithm will be provided in Section 3.

2.3 Approach and Objectives

Summarizing, the goals of this work are:

- model the data center architecture as a distributed system and assign its management to a decentralized system built on top of the SelLet architecture;
- define a proper bio-inspired algorithm that may be run in such system to obtain actual energy savings;
- define the integration of such algorithm into the self-organizing architecture of the data center pointing out the integration issues that have arisen and that are common when dealing with such type of algorithms;
- run the proposed algorithm in a simulation environment to evaluate its performance, and compare the results with theoretical optimal solutions.

3 The Model

In this section we show the model of the system from an architectural and an algorithmic point of view.

3.1 Architectural Model

From the architectural point of view the system is composed of a set of physical servers interconnected to each other: this is the physical layer of the system. Each server may run virtual machines and is supported by a self-organizing component called SelfLet [2, 3], in charge of executing self-organizing behavior.

Server. The server is the entity in charge of computation of the normal load in a data center. In addition to that, each server interacts with the scouts, which are located on it time by time.

In this paper, we assume a simplified but realistic server architecture. Nowadays many data centers have undergone a process of server consolidation in order to reduce the number of physical machines. Such a process involves the introduction of a software layer called Virtual Machine Monitor (VMM), which is a system providing an environment to support an operating system, and its applications and processes. Figure 2 reports the description of a single server machine, which is able to run three different kinds of servers (e.g., they could be a database server, a web server, and an application server), running two different operating systems and their own sets of applications. Such a configuration allows considering servers as applications, which can be moved and run by another machine equipped with an appropriate VMM.

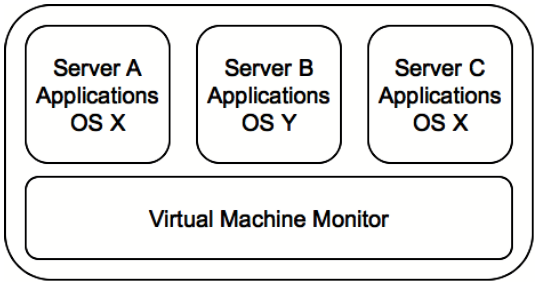


Fig. 2. Schema of a consolidated server.

Servers are linked to each other as in a normal data center and each one has the following set of characteristics:

- *Load*: each physical server contains a list of running virtual machines (VMs), which can be migrated from a server to another. In particular, VMs in a single physical server are logically grouped into packages, which are the actual object of migration. The dimension of the package can be customized depending on the characteristics of the physical machines.
- *Capacity*: it is the maximum amount of resources available on the server. In the current model it is considered as a single parameter grouping together different characteristics like local memory, CPU computational power, etc.

– *Power consumption class*: it is a value indicating the technology used to implement the server and is considered as an indicator of power consumption. Power consumption class is an integer value spanning between 1 (most power saving machines) and 4 (most power consuming machines).
– *State*: the status of a server at a certain moment can be *on*, i.e. it is normally computing its load, and *hibernated*, i.e. the machine is put into a particular power saving mode because it has moved all its VMs.

The data center is modeled using a logical classification based upon layers, similarly to what is described in [15]. Servers in the model can be hierarchically organized in: (*i*) a storage network area, i.e., a set of machines that share storage devices and that have a small cost of VM migration, (*ii*) cluster, i.e. a set of machines logically grouped together because they are linked to the same switch or are managed by the same load balancer, and (*iii*) data center level, i.e., all servers that belong to a given data center and are located in the same physical site. As it can be noticed from Figure 5, this hierarchical structure is not mandatory and depends on the particular data center (for instance, a small data center could not need the concept of cluster), nevertheless, it is useful in the attempt to model real world scenarios, to consider heterogeneous configurations (e.g., bandwidth differences among clusters), and to model constraints for VMs migration (e.g., not every server could be able to run all the VMs).

SelfLet. The autonomic architecture of this paper is based on the concept of SelfLet, which can be defined as an autonomic architectural component able to dynamically change and adapt its internal behavior according to modifications in the environment. A SelfLet is also able to interact and cooperate with other SelfLets to achieve high-level goals.

SelfLets have the following characteristics: they are identified univocally by an ID and can be defined as belonging to one or more types, which is an application-dependent property. At any given time a SelfLet can also belong to a Group, constituted by its neighboring SelfLets, to which communication can take place. A SelfLet executes one or more possible Behaviors. A Behavior can be seen as a workflow that involves some actions such as the invocation of Abilities, i.e. services, which can be local or remote. A Behavior usually results in the fulfillment of one or more Goals. Possible changes in the internal state of the SelfLet or in the environment may trigger particular intelligent rules, called Autonomic Rules, which may modify the Behavior, and add/remove/run Abilities. Specific kinds of Abilities are those that perform some autonomic tasks, for example, they can create an aggregation of neighbors respecting certain properties.

Figure 3 represents the internal architecture of a SelfLet. The communication with the other SelfLets takes place thanks to the Message Handler. The Negotiation Manager is in charge of publishing the availability of achieving certain Goals in a SelfLet or of requesting the achievement of some Goal to other SelfLets if needed. The Behavior Manager is the one that executes Behaviors while all Abilities are managed by the Ability Execution Environment: it offers the

primitives to install, store, and uninstall Abilities without restarting the SelfLet. The Internal Knowledge is basically an internal repository, which can be used to store and retrieve any kind of information, needed by any of the SelfLet components. Finally, the Autonomic Manager has the task of monitoring the SelfLet and its neighbors and of dynamically adjusting the SelfLet according to a given policy by firing Autonomic Rules.

In our work we plan to instrument the SelfLet by expressing the self-organization algorithm as SelfLet Behaviors and Autonomic Rules, and by implementing the interfaces for monitoring server parameters and for starting VM migrations as SelfLet Abilities.

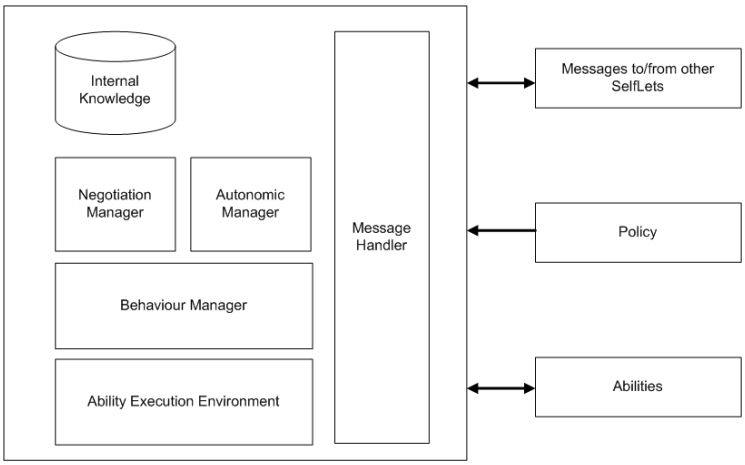


Fig. 3. Architecture of a SelfLet.

3.2 Algorithm Model

From the algorithmic point of view the model we are proposing takes some elements from current research in the field of biologically inspired self-organization algorithms, in particular from collective decision making in animals colonies. Nature offers several examples of groups of mammals and birds, which are able to behave collectively and take all together some decisions like whether to migrate or not from a site to another, whether to stay or leave a group, etc. The case of migration in some insects is characterized by the following behaviors:

- Some individuals, called scouts, leave their groups and go looking for different and better sites.
- These individuals evaluate new sites comparing them with their origin sites. When they go back home, they can decide to convince the rest of the group to start a migration.

The algorithmic model presented in this work is based on the same principle. There are three types of entities:

- the *community of VMs* (i.e., the colony in biological terms) residing on a certain server;
- the *scouts*, which explore different physical servers and compare them with the origin one;
- the *server managers*, which permanently reside on each physical machine, and decide to move a community of VMs based on the information brought back by scouts.

When scouts find a better place for the community, they notify the server manager that will then make the decision whether to migrate the VMs to the new site.

Scout. The scout is the entity in charge of investigating the data center characteristics and sharing them among the servers. This information is used to make decisions about migrations of load among the servers. A scout is created in all the physical nodes by server managers during the algorithm initialization and, differently from the original algorithm presented in [14], it does not belong to the server, but it cruises the data center following the links among the physical machines. During the exploration each scout queries the server managers to know the actual configuration of the current location. The following list reports the main characteristics of a scout:

- Current location: a scout can be located only on one server at a given time. The location is identified by the server identification number, which could be its IP address.
- Lifetime: a scout has a parameter indicating the maximum length of its life. At the current state of the model, this value is set to infinite, indicating that scouts never die, but a finite value could be useful to limit the number of homeless scouts in the data center in case of a server failure or crash.
- List of preferred servers: each scout stores the information obtained during the data center investigation into a memory, which classifies the visited servers according to two parameters:
  - *power consumption class*: this is the first characteristic to be evaluated. In our case a server with a lower benchmark data consumption is preferred;
  - *amount of free resources*: this is the second issue to be evaluated. In our case the server with more unused resources is preferred.

The way a scout works is illustrated by Algorithm 1: during each iteration a scout performs the task of monitoring and storing the information of the visited server and then moves to another server. Figure 4 shows a configuration in which servers are represented by large grey circles, scouts and VMs are depicted with triangles and diamonds respectively. Links among servers are the only way scouts can use to move from a server to another for the investigation.

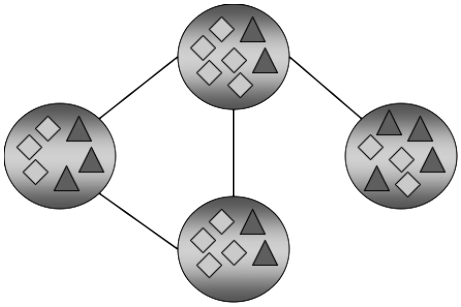


Fig. 4. Example of a small data center configuration on a random network.

Algorithm 1 Scout Thread

```
parameters: preferenceList
while TRUE do
  scout.preferenceList.update(current_node.Id,current_node.Status)
  scout.move()
end while
```

Server Manager. The Server Manager is the entity in charge of communicating with the scouts and of taking decisions about movements of VMs. Decisions on moving or not a VM package is made according to the following parameters:

- Server choice function: server managers could be in the situation of migrating a VM package while two or more scouts are indicating different possible servers. In this case the order of evaluation for the destination machine by the server manager is the following:
  1. a server with the lowest power consumption class is preferred;
  2. the server that would offer the highest percentage of use of resources is preferred;
  3. in case the previous parameters are equal, the destination node will be chosen randomly.

  These criteria are fundamental to ensure the system convergence to a stable state, and to avoid continuous switches of servers between states.
- Migration probability: as in many biologically inspired systems, decisions whether to migrate or not a VM package is not deterministic, but follows a probability distribution such as the one showed in Figure 6, which illustrates how the probability of not migrating VMs increases with the load percentage. This non-determinism is needed to avoid situations in which a loaded server goes on migrating all its VMs, with the result of increasing network traffic without improving significantly the global power saving.
- Migration inhibitors: previous choice criteria are not the only reasons to avoid migrations from a given server. In particular there are two other reasons why VMs could be locked in a server: (i) the server manager has a

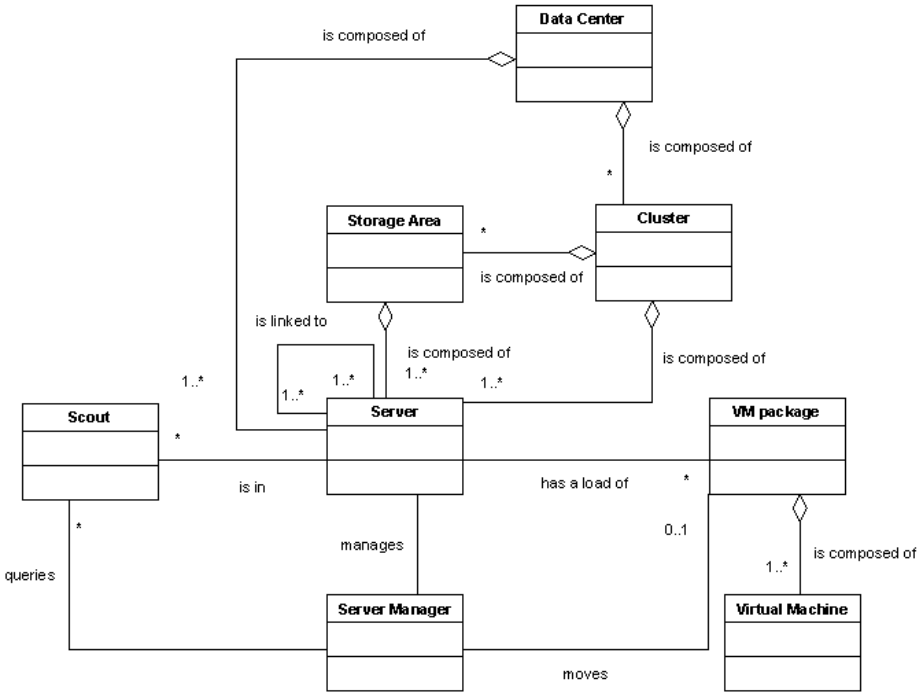


Fig. 5. Conceptual model of the data center.

flag variable that indicates that a given set of VMs are not available to migrate, e.g. because they are running important processes and cannot be stopped, or because of some QoS constraints; and (*ii*) a given VM cannot migrate immediately after its arrival, but the server manager has to wait some time intervals before moving a VM that has just arrived. This behavior has been introduced to avoid bandwidth overload along with its related power consumption, and also vicious circles that would make some VMs move continuously, preventing them from running their jobs.

The way a server manager works is illustrated in Algorithm 2.

4 Integration

The SelfLet framework offers an environment suitable to develop and deploy the algorithm of Section 3.2.

We have defined a two layered conceptual architecture that extends the model of a server depicted in Figure 2. This architecture is showed in Figure 7. The *Physical layer* is composed of the physical devices of a data center; for the sake of simplicity we are considering mainly computer machines and a network

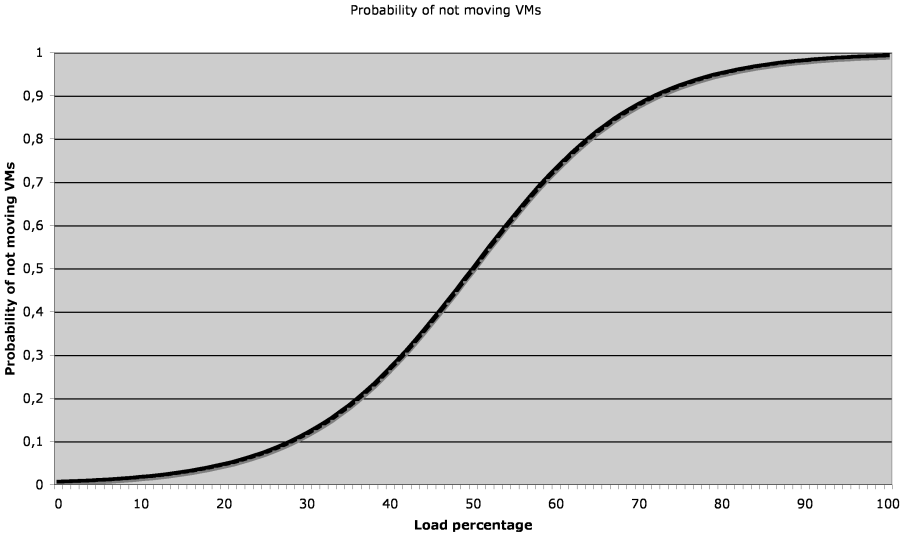


Fig. 6. Probability distribution used to choose if a set of VMs should be moved, depending on the current total load of the machine.

communication system. The *Application layer* is, in turn, the virtualization level, and it is composed of virtual machines, which can be run by a single physical server. The *SelfLet layer* is composed of SelfLets, which are in charge of managing the autonomic behavior of server machines and their VMs. It should be noticed that the SelfLet and the Application layers are designed at the same logical level as they both need to interact with the Physical Layer.

Communication among the three modules occurs in the following way:

- the communication between the Application layer and the Physical layer is necessary to allow the execution of VMs, which are managed by the VMM;
- the communication between the Application layer and the SelfLet layer allows the latter to know all the properties of each VM, such as its number, and the possibility to move it;
- the communication between the SelfLet layer and the Physical layer is needed to know the global status of the machine, such as power consumption class, status of hibernation, and resources utilization, indeed the modular structure of a consolidated server keeps each VM completely independent from the others, and a single VM is not able to obtain information from the others. Moreover, the SelfLet is in charge of deciding whether to switch the status of the physical machine to hibernation according to the system constraints.

To satisfy the needs of the presented algorithm, SelfLet's modules will be used the following way:

Algorithm 2 Server Manager Thread

```
parameters: preferenceList, knownScouts, VMSet, hibernated
scout=new Scout
scout.move()
while TRUE do
  if !current_node.isLocked() then
    movableVMs=getMovableVMs(VMSet)
    if movableVMs.notEmpty() then
      for scout in knownScouts do
        preferenceList.update(scout(i).queryPreferenceList())
      end for
      bestLocation=evalLocation(server.preferenceList)
      if bestLocation.notEmpty() then
        movableVMs.move(bestLocation)
      end if
    end if
    if VMSet.notEmpty() then
      hibernated=false
    else
      hibernated=true
    end if
  end if
end while
```

- the Behavior Manager is in charge of managing the actual Behavior of each server;
- the Autonomic Manager has the task of monitoring all the other components (and the communications among them) and of firing Autonomic Rules, which may be triggered to dynamically adjust the SelfLet according to a given policy: for example it can take the decision of locking the server status to avoid VMs migrations in particular situations and whether to move a VM package;
- the Internal Knowledge repository can store information on the network learned from moving scouts (forwarded by the SelfLets of other physical servers), such as other servers' status;
- moreover, SelfLets need a set of additional customized Abilities to be able to manage the system:
  - the Scout Management Ability provides the primitives for the SelfLet to create/destroy new scouts, and receive/forward external scouts. This is done using the communication primitives provided by the physical layer. One of the most important benefits of this Ability is the update of the internal knowledge that may then be used by the monitors described below to compute metrics.
  - the Server Interface Ability allows the information exchanges between the Behavior Manager and the Physical layer;

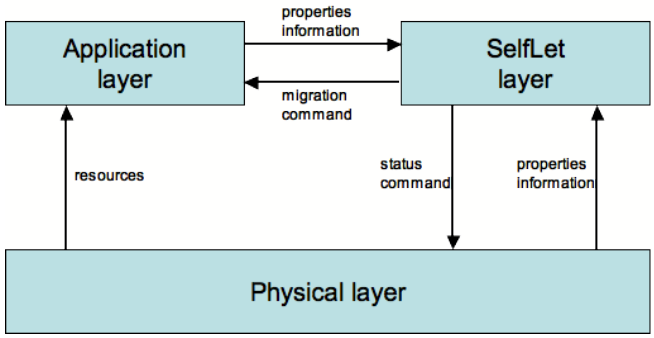


Fig. 7. Layered architecture of the system.

- the Server Monitor Ability has the task of monitoring the status of the server to calculate metrics on server resource utilization and server characteristics such as energy class.
- the Application Interface Ability has the task of managing the information exchanges between the SelfLet and the Application layer;
- the Application Monitor Ability, similarly to the Server Monitor, is in charge of computing metrics on virtual machines or querying for their characteristics, e.g. the kind of server (database server, web server, etc), statistics on utilization rate and resources requirements.

Figure 8 represents the Behavior of a SelfLet. Basically, it is composed of three states: *Active with load*, i.e. the server is working and the SelfLet is communicating with other SelfLets sending and receiving load and scouts; *Active without load*, which is similar to the previous state but the server has not a current set of VMs and is kept on for some constraints or policies set by the datacenter administrator; finally, the SelfLet can turn to *Hibernated* if there is no constraint and it has no load. Autonomic Rules allow the transitions from a state to the next one. These rules are based on the SelfLet's activities. They analyze the status of the server and allow the Autonomic Manager to fire the rules.

Summarizing, the SelfLet model with the proposed extensions has the proper Abilities to communicate and collect data from the other modules of the system (Physical layer and Application layer), to create, send, receive, or destroy scouts that move through the system using diffusive communication algorithms, and to store and elaborate the data provided by moving scouts. These data are stored in the Internal Knowledge of the system and is used to fire rules that may result in decisions whether to move a VM or not. All the steps of the algorithm are codified as a workflow that is managed by the Behavior manager and that may be sent to other SelfLets (along with the Autonomic Rules) to trigger the same algorithm on neighboring SelfLets in an epidemic way. This epidemic spread is also useful in case of future algorithm improvements, since there is no need to

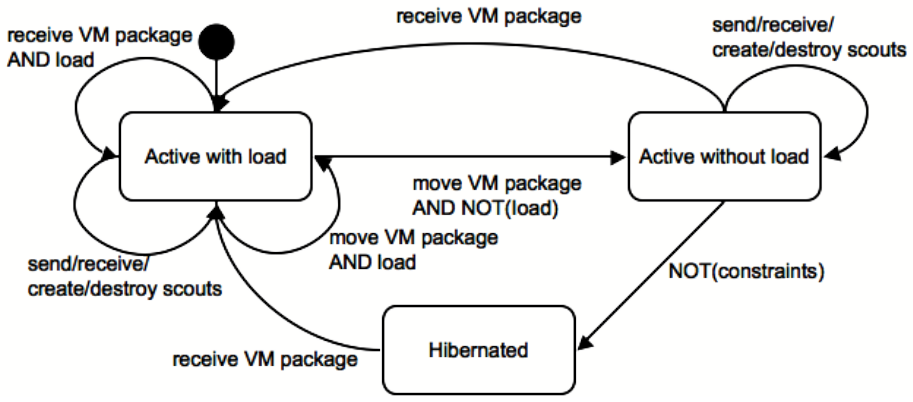


Fig. 8. Behavior of a SelfLet.

manually change the configuration of each SelfLet. Moreover the Abilities that are required by the algorithm may be automatically migrated from a SelfLet to the other as needed in a seamless way, thus reducing system management costs.

5 Simulations

This section shows some preliminary results that have been obtained from a prototype implementation of the power-saving self-organization algorithm proposed in Section 3.2. The prototype has been implemented using the PeerSim tool [16], which is a peer-to-peer network simulator written in Java, which allows simulations of systems with varying cardinality. The simulation engine that has been used divides the simulation into steps: each step corresponds to atomic operations that involve all the nodes of the systems at the same time. Due to this simplification, these steps cannot be seen as a timing measure, however in this preliminary state, simulation steps are actually the best indicator for studying the algorithm convergence in different situations.

Results obtained by the implemented model have been compared with results obtained on the same network by a static global optimizer based on Java wrapped LpSolve operations research library [17]. This optimizer computes the best possible optimization case by case, spreading the whole network load to the most energy efficient machines, and using the 100% of their resources. Therefore, it is used as an upper limit to understand the performance of the proposed algorithm. The indicators that have been used to evaluate the algorithm's performance are the following:

- *Power consumption*: calculated as the sum of the energy class indicators of each server.
- *Number of hibernated servers*: number of servers that have no VMs at a certain simulation step.

– *Average server usage*: the mean of each server usage percentage.
– *Power saving percentage*: reduction of power consumption with respect to the initial situation.
– *Number of steps to reach a stable state*: the number of steps needed for the data center to reach a stable configuration of hibernated servers.

The remainder of this section shows the results obtained according to the following simulation plan. In the first simulation we report the results of some experiments made to understand the behavior of the algorithm when *varying the number of scouts per each server*. In the following simulations we evaluated the algorithm's performance when *changing the initial load* for each server and the total number of servers. After several runs we have seen that the results obtained in all the experiments stayed coherent with an average discrepancy below 10%.

Varying the number of scouts. The first set of simulations focused on understanding the behavior of the system when varying the initial number of scouts per each node. Simulations confirm our initial hypothesis, since when increasing the number of scouts the final performance is better. Anyway there is a threshold over which system's performance does not increase significantly when such limit is reached, therefore a higher number of scouts does not produce further increase in final savings. For this evaluation the network was configured using 1000 servers with an initial average load of 15%. Results of these simulations are shown in Figure 9: the three subfigures show respectively the total number of hibernated servers, the total power consumption expressed as the sum of the power consumption index of each server, and the average server usage, vs the total number of scouts. From these results we can see that the best performance is obtained by the configuration with 6 initial scouts per server. On the other hand, Table 1 shows how network load, which is strongly dominated by the number of VM migrations, gets higher when increasing the number of scouts. Therefore, it is important to establish the number of scouts that plays the role of threshold (6 in the example), so to avoid an increase of the network load without introducing additional energy savings.

Table 1. Number of VM migrations after the algorithm has reached a stable state.

Number of scouts	**Number of migrations**
1	8198
3	11510
6	12266
9	13286
12	13458

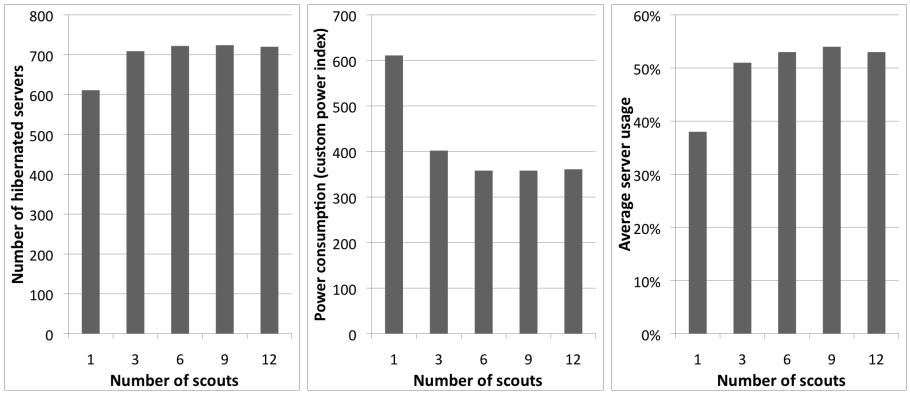


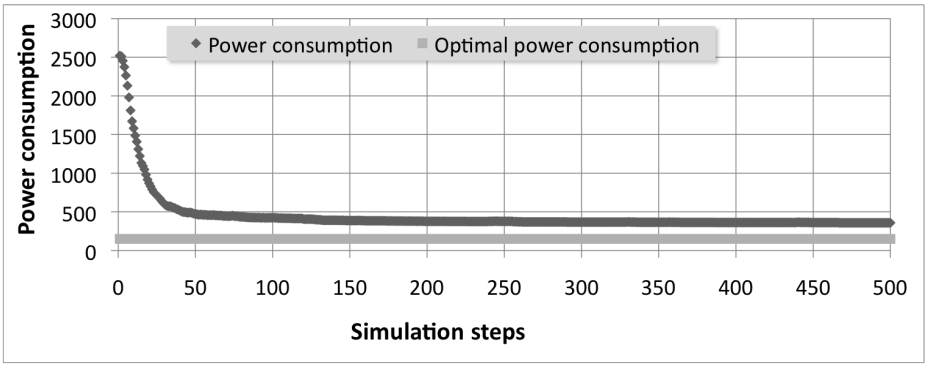
Fig. 9. Comparison of the behavior of the algorithm with a different number of scouts.

Experiments with 15% of initial load. For this simulation we have chosen an initial load for the servers equal to 15% of their capability, to observe how the algorithm is able to perform when conditions do not give narrow margins. Figure 10a shows the behavior of the algorithms in terms of power consumption: the system reaches a stable state after 300 simulation steps and gives an 88% saving. Table 2 shows also the number of hibernated servers and the average server usage (in stable conditions) for different network sizes.

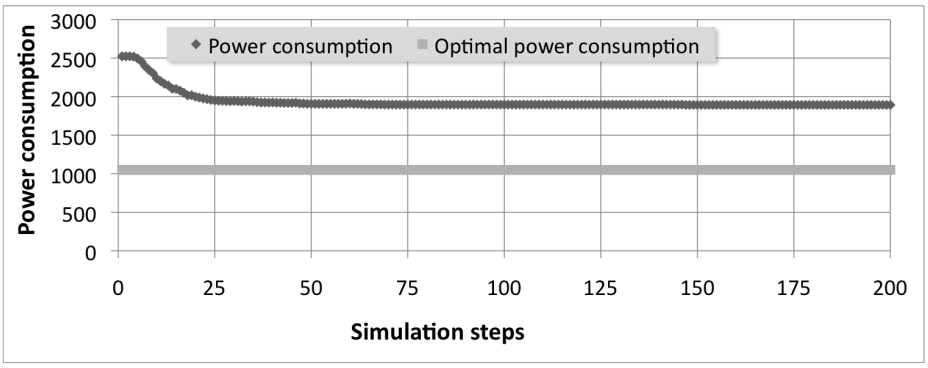
Experiments with 60% of initial load. In this experiment we investigate the system's behavior when servers' initial load is 60%. Figure 10b shows the behavior of the algorithms in terms of power consumption. The system reaches a stable state after 80 simulation steps and gives a 25% saving, but leading servers to a 71% usage of their resources, as shown in Table 2.

Experiments with 70% of initial load. Similarly to previous experiments, in this simulation our goal is to explore the system's behavior in a sort of limit condition. In this case the average initial server load is 70% and there is not a great range of actions that can be carried out to further improve power saving since the algorithm has been tuned not to overload the server and reaching a desired 80% load. Figure 10c shows the power consumption curve: in this case the algorithm obtains a 10% saving in about 50 simulation steps. From Table 2 we can also see that the algorithm manages to hibernate 7% of servers and to increase the average load up to 75%.

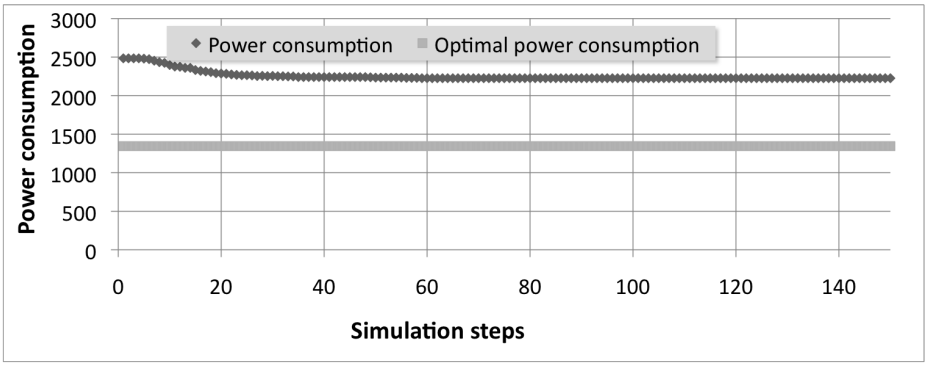
Discussion. As previously discussed, the presented approach obtains very good results when the whole load of the data center is low, reaching a percentage of saving up to 88%. In the other situations the results are still good even if the conditions are worse because the servers are saturated. In all the situations



(a) 15% load



(b) 60% load



(c) 70% load

Fig. 10. Comparisons of the power consumption between the presented algorithm (dark dots) and the ideal optimal solution (light-gray line) when varying the initial load.

the algorithm has also shown a great capability to scale with respect to the number of servers, but with a different behavior in presence of different loads:

Table 2. Results comparisons.

N. servers	Initial avg load	Power saving	Hibernated servers	Final avg load
100	15%	82%	69%	48%
500	15%	85%	72%	52%
1000	15%	88%	72%	53%
100	60%	31%	21%	75%
500	60%	25%	17%	71%
1000	60%	25%	17%	71%
100	70%	5%	3%	72%
500	70%	10%	7%	74%
1000	70%	10%	7%	75%

this is probably due to the higher number of alternatives for the load balancing. Another advantage is the fact that the self-organizing logic is scattered across all the nodes, thus increasing the reliability with respect to a centralized solution. Finally the simulation steps needed to reach a stable state can be remarkably low. In less loaded networks stability is reached in about 30-40 steps, but even in larger networks the convergence has a similar rate.

The main drawback of the approach is the increased amount of traffic overhead in the network due to VM migrations. The actual simulations do not show this overhead since each message exchanged is an atomic operation that takes one simulation step, however some parameters that may affect overhead are the size of scout information, the size of each VM, and the number of scouts per each server. Decisions need to be calibrated based on this information to find a trade-off value between power saving efficiency and the amount of network overhead, therefore new and smarter policies to bound the number of migrations should be introduced. Moreover, it is clear that having a high capacity communication network layer among servers become a necessary condition, thus representing the most critical limitation of this work.

6 Related Work

This section contains an overview of the state of the art of existing works that are related to the approaches we have used in this paper. The section is organized as follows: Section 6.1 shows existing approaches for optimizing power consumption in data centers, Section 6.2 shows an overview of the field of bio-inspired self-organization, and finally Section 6.3 presents some alternative architectures that may be integrated into existing systems with the purpose of running bio-inspired self-organization algorithms. Although our results are preliminary, they can be considered indicative of the contribution that our work gives. Indeed our results are comparable to Muse [18], which is able to reduce server energy consumption by 29%-78% for representative Web workloads.

6.1 Approaches for Power Optimization in Data Centers

The typical approaches in literature are divided into two main classes: centralized and distributed.

Centralized Approaches. These approaches are the oldest ones and the ones able to give results that are the closest to the optimum; however having a central element creates coordination and fault-tolerance problems in presence of a very high number of servers. In this context our approach is able to address this limitation, but with the drawback of having a less optimal final result when compared to centralized approaches. Hereafter, we shortly describe some of these works.

In [19, 20] the authors have tried to identify workload time series to dynamically regulate CPU power and frequency to optimize power consumption. This attempt led to a reduction of consumption by to up 36%.

Also the work presented by Pinheiro *et al.* [21] addresses the problem of power consumption in server clusters. In this case the objective is turning off the highest number of servers making a redistribution of the load, taking into account a maximum degradation of performance. In this case a machine is in charge to choose which server should be turned on and off per each cluster of the network, being able to measure the overall performance.

Another centralized approach is introduced by Chase *et al.* [18]. In this case an economic perspective is taken to manage shared service resources in hosting centers. In the presented ad-hoc solution, the Muse operating system, a central unit called *Executive* takes requests as inputs and takes resource allocation decisions, while other three modules are in charge of: (*i*) monitoring and estimating the load, (*ii*) switching incoming traffic to the selected server in a transparent way for the services and (*iii*) managing a pool of generic and interchangeable resources. The economic similarity is due to a utility function that maximizes "revenues".

In [22], Elnozahy *et al.* present five different policies to manage energy saving in a server cluster environment. This work (and the one presented in [21]) aims at optimizing voltage scaling by dynamically adjusting the operating voltage and frequency of the CPU to match the intensity of the workload.

The work shown in [23] by White and Abels presents a computing model called VDC in which a centralized dynamic control system is in charge of managing resources for the applications, which are seen as isolated virtual machines. In this approach the resources are virtualized so that they can be provisioned and resized dynamically to applications as required. Moreover applications may frequently migrate from one set of resources to another. VDC implements applications as isolated virtual machines, which are built on top of their virtualized resources (server, storage, and switches), and under their virtualized service layer (user and service levels) across the entire data center.

Distributed Approaches These approaches aim to solve the power saving problem in a distributed way, so that they may scale well in presence of large-

scale system. The problem of these approaches is that they are often based on networks that are usually organized as a hierarchy or as a lattice. This solves scalability and fault-tolerance problems, but has the drawback of being difficult to maintain in presence of very dynamic situations since hierarchies/matrices are difficult to be reconstructed when they are broken. In our approach every node of the network is equal to each other and there are no hierarchies: this means that there are almost no penalties if nodes are added/removed or they change their status (e.g., from hibernated to active) in a fast and unpredictable way. The following are some of the existing distributed approaches.

The work done by Khargharia *et al.* in [24] presents a theoretical framework that optimizes both power and performance for data centers at runtime. In particular the approach is based on a local management of small clusters and on a hierarchy to manage power. The hierarchy is composed of three levels: (*i*) cluster, (*ii*) server, and (*iii*) device, each one with a fine granularity because it is able to address consumption of processor, memory, network and disks.

Bennani and Menasce [25] present a similar hierarchical approach with respect to [24], addressing the problem of dynamically redeploying servers in a continuously varying workload scenario. In this case servers are grouped according to an application environmental logic, and a so-called *local controller* that is in charge of managing a set of servers. In turn, all the local controllers report the workload prediction and the value of a local utility function to the global controller, which takes decisions on the collective resource allocation.

Das *et al.* [5] present a multi-agent system approach to the problem of green performance in data center. As for aforementioned papers, the framework is based on a hierarchy, according to which a resource arbiter assigns resources to the application managers, which, in turn, are in charge of managing physical servers.

6.2 Bio-inspired Self-organization Algorithms

Bio-inspired algorithms are self-organization algorithms that share some principles with phenomena of the natural world [1]. This kind of biological behaviors are often taken as inspiration to develop new algorithms because they are usually built upon very simple elements that are instructed with very simple rules. Moreover each element has very limited memory and computational capabilities. The advantages of these algorithms are the fact that the complexity is spread across all the nodes of the system, and that, if an element is removed from the system or behaves in an unpredictable way, the system keeps working without problems. These algorithms are often characterized by some probabilistic behaviors that are needed to explore the environment (and the possible solutions) and have the chance of finding (or improving) their goal [26, 27]. Some studies of these principles have been proposed in [28, 29]: these works clearly show these advantages. The main drawbacks of bio-inspired techniques lie in the fact that they are difficult to implement, study and calibrate in real systems. Furthermore they are not able to obtain good results in presence of static environments with

a small number of elements. This happens because in this way the chances of finding a good solution by randomly exploring the solution space are much lower.

An example of algorithm that uses the migration principle has been proposed by Saffre in [14]. Our work shows how such algorithm that has been previously described using a toy example (workers, scouts and sites are expressed in generic terms in the previous work) may be modified and tailored to tackle a concrete problem such as the power optimization problem. Other studies that propose some integration principles to use bio-inspired techniques in existing systems have been proposed in [30, 31].

6.3 Distributed Self-organization architectures for Bio-inspired Algorithms

In this work we propose to add to the existing datacenter architecture a self-organizing framework (the SelfLet [2] architecture). We chose to use it because it has been developed with the idea of decentralization and all other bio-inspired principles in mind, so it is the natural complement for this class of bio-inspired algorithms. Beside that, most of existing self-organizing architectures – included the ones based on the IBM Autonomic model [12] (although with some more effort) – may be tailored to execute such algorithms.

Other self-organizing architectures include Autonomia [32], which uses a two layer approach, where the first contains the execution environment and the other manages the resources.

AutoMate [33], which has a multi-layered architecture optimized for scalable environments such as decentralized middleware and peer-to-peer applications.

CASCADAS Toolkit [34], which has been designed to have runtime changing goals, plans, rules, services and a behavior modeled as a finite state machine that may change overtime.

BIONETS [35] is an architecture based on two elements: tiny nodes (T-nodes, characterized by limited computational and communicational capacity like sensors and tags) and user nodes (U-nodes, that are terminal user nodes such as laptops and phones, and have full communication and computational capacity). U-nodes host services and interact with the environment through the T-nodes in their proximity, while U-nodes can also communicate between themselves to exchange information. Anthill [36] is a self-organizing architecture explicitly based on the analogy with ant colonies: it is composed of a network of interconnected nests (peers), which handle requests originated by local users by creating ants, autonomous agents that perform their tasks by traveling across the network. In this context the ants are the autonomic elements of the system and can be modified and evolve to better respond to the requested service.

In conclusion choosing the most suitable self-organization architecture depends on the deployment scenario and on the type of the existing system.

7 Conclusions and Future Work

In this work we have shown a possible way to exploit the self-* properties provided by a bio-inspired algorithm in a self-organizing architecture through their application to a real world use case scenario related to Green IT. In particular, it has been widely discussed how modern data centers are continuously growing up and how their dimensions represent a serious risk for their future development and management. Therefore a new bio-inspired algorithm has been introduced, based on the idea of colonies migrations. The advantages of the presented algorithm are mainly related to its self-management characteristics and to its ability to react autonomously to changes in the environment and in the requirements.

We have also shown how to benefit from these advantages by integrating this algorithm into the architecture of a modern data center composed of physical nodes, which run virtualized machines on top of them with the support of the SelfLet self-organizing framework. The only primitives that the whole system relies on are the possibility to move a virtual machine from a node to another one, and the possibility to hibernate idle machines and resume the operation of hibernated machines if the available ones are not able to deal with the actual workload.

The performance of the presented approach is pretty satisfying. Experiments have shown significant savings in power consumption through a technique based on the hibernation of the highest possible number of servers, preferably among high-consumption ones.

Some limits of the current study that will be further investigated in a future work are the following: (i) introduction of mechanisms to limit the number of VMs migrations, to let tasks run and to decrease the network load; (ii) further investigation on the scouts' life-cycle: in a real world scenario, in which servers can also crash and some scouts can be lost, it could be possible to regenerate them; (iii) further investigation on the scout memory to decide the number of sites to be remembered; (iv) model refinement to introduce QoS constraints and policies to understand if the system is able to respect Service Level Agreements in a changing environment.

In conclusion, from a methodological point of view this paper has shown a clear example of integration of a self-organization algorithmic logic into an architecture, which is supposed to support self-organizing system management. Moreover we have put some evidence and given some clues on the utility of bio-inspired approaches in a raising and important issue like power saving in data centers.

Acknowledgements. This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227077-SMScom.

References

1. Camazine, S., al.: Self-organization in biological systems. Princeton University Press, Princeton (2001)

2. Devescovi, D., Di Nitto, E., Dubois, D.J., Mirandola, R.: Self-Organization Algorithms for Autonomic Systems in the SelfLet Approach. In: Autonomics, ICST (2007)
3. Bindelli, S., Di Nitto, E., Furia, C., Rossi, M.: Using Compositionality to Formally Model and Analyze Systems Built of a High Number of Components. In: 15th IEEE International Conference on Engineering of Complex Computer Systems, IEEE Computer Society (2010)
4. Capra, E., Merlo, F.: Green IT: Everything strarts from the software. In: ECIS '09: Proceedings of the 17th European Conference on Information Systems. (2009)
5. Das, R., Kephart, J.O., Lefurgy, C., Tesauro, G., Levine, D.W., Chan, H.: Autonomic multi-agent management of power and performance in data centers. In: AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems (2008) 107–114
6. Murugesan, S.: Harnessing green IT: Principles and practices. IT Professional **10**(1) (2008) 24–33
7. Kumar, R.: Important power, cooling and green IT concerns. Technical report, Gartner (January 2007)
8. Brown, E., Lee, C.: Topic overview: Green IT. Technical report, Forrester Research (November 2007)
9. Josselyin, S., Dillon, B., Nakamura, M., Arora, R., Lorenz, S., Meyer, T., Maceska, R., , Fernandez, L.: Worldwide and regional server 2006-2010 forecast. Technical report, IDC (November 2006)
10. Lamb, J.: The Greening of IT: How Companies Can Make a Difference for the Environment. IBM Press (2009)
11. Kurp, P.: Green computing. Commun. ACM **51**(10) (2008) 11–13
12. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
13. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Book (2004)
14. Saffre, F., Tateson, R., Marrow, P., Halloy, J., Deneurbourg, J.L.: Rule-based modules for collective decision-making using autonomous unit rules and inter-unit communication. Technical report, Deliverable 3.5 - IP CASCADAS (2008)
15. Greenberg, A., Hamilton, J., Maltz, D.A., Patel, P.: The cost of a cloud: research problems in data center networks. SIGCOMM Comput. Commun. Rev. **39**(1) (2009) 68–73
16. Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: The Peersim simulator `http://peersim.sf.net`.
17. : LpSolve, a Mixed Integer Linear Programming (MILP) solver. Available online at http://lpsolve.sourceforge.net
18. Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M., Doyle, R.P.: Managing energy and server resources in hosting centers. SIGOPS Oper. Syst. Rev. **35**(5) (2001) 103–116
19. Bohrer, P., Elnozahy, E.N., Keller, T., Kistler, M., Lefurgy, C., McDowell, C., Rajamony, R.: The case for power management in web servers. Kluwer Academic Publishers, Norwell, MA, USA (2002)
20. Lefurgy, C., Rajamani, K., Rawson, F., Felter, W., Kistler, M., Keller, T.W.: Energy management for commercial servers. Computer **36**(12) (2003) 39–48
21. Pinheiro, E., Bianchini, R., Carrera, E.V., Heath, T.: Load balancing and unbalancing for power and performance in cluster–based systems. In: In Proceedings of the Workshop on Compilers and Operating Systems for Low Power. (2001)

22. Elnozahy, E.N.M., Kistler, M., Rajamony, R.: Energy-efficient server clusters. Power-Aware Computer Systems **2325** (January 2003) 179–197
23. White, R., Abels, T.: Energy resource management in the virtual data center. In: ISEE '04: Proceedings of the International Symposium on Electronics and the Environment, Washington, DC, USA, IEEE Computer Society (2004) 112–116
24. Khargharia, B., Hariri, S., Yousif, M.S.: Autonomic power and performance management for computing systems. Cluster Computing **11**(2) (December 2007) 167–181
25. Bennani, M., Menasce, D.: Resource allocation for autonomic data centers using analytic performance models. In: Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on. (June 2005) 229–240
26. Pasteels, J., Deneubourg, J., Goss, S.: Self-organization mechanisms in ant societies (i) : trail recruitment to newly discovered food sources. In: From individual to collective behavior in social insects, Birkhaser, Basel, Eds J.M. Pasteels & J.L. Deneubourg. Experientia Supplementum (1987) 54, 155–175
27. Nicolis, S., al.: Optimality of collective choices: a stochastic approach. Bulletin of Mathematical Biology (2003) 65, 795–808
28. Babaoglu, O., Canright, G., Deutsch, A., Caro, G.A.D., Ducatelle, F., Gambardella, L.M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., Urnes, T.: Design patterns from biology for distributed computing. ACM Trans. Auton. Adapt. Syst. **1**(1) (2006) 26–66
29. di Nitto, E., Dubois, D.J., Mirandola, R.: On exploiting decentralized bio-inspired self-organization algorithms to develop real systems. Software Engineering for Adaptive and Self-Managing Systems, International Workshop on **0** (2009) 68–75
30. Serugendo, G.D.M., Karageorgos, A., Rana, O.F., Zambonelli, F., eds.: Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering. In Serugendo, G.D.M., Karageorgos, A., Rana, O.F., Zambonelli, F., eds.: Engineering Self-Organising Systems. Volume 2977 of LNCS., Springer (2004)
31. Nakano, T., Suda, T.: Applying biological principles to designs of network services. Appl. Soft Comput. **7**(3) (2007) 870–878
32. Hariri, X.D., Xue, S.L., Chen, H., Zhang, M., Pavuluri, S., Rao, S.: Autonomia: an autonomic computing environment. In: IEEE International Performance, Computing, and Communications Conference, 2003. (2003)
33. Parashar, M., Liu, H., Li, Z., Matossian, V., Schmidt, C., Zhang, G., Hariri, S.: Automate: Enabling autonomic applications on the grid. Cluster Computing **9**(2) (2006) 161–174
34. Hoefig, E., Wuest, B., Benko, B.K., Mannella, A., Mamei, M., Di Nitto, E.: On concepts for autonomic communication elements. In: International Workshop on Modelling Autonomic Communications. (2006)
35. De Pellegrini, F., Miorandi, D., Linner, D., Bacsardi, L., Moiso, C.: Bionets architecture: from networks to serworks. In: Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd. (Dec. 2007) 255–262
36. Babaoglu, O., Meling, H., Montresor, A.: Anthill: A framework for the development of agent-based peer-to-peer systems. Distributed Computing Systems, International Conference on **0** (2002) 15