A Self-Organized Load-Balancing Algorithm for Overlay-Based Decentralized Service Networks*

Giuseppe Valetto†, Paul L. Snyder†, Daniel J. Dubois‡, Elisabetta Di Nitto‡, and Nicolò M. Calcavecchia‡
†Drexel University, Department of Computer Science, Philadelphia, Pennsylvania, USA
‡Politecnico di Milano, Dipartimento di Elettronica e Informazione, Milan, Italy
valetto@cs.drexel.edu, plsnyder@drexel.edu, dubois@elet.polimi.it, dinitto@elet.polimi.it, calcavecchia@elet.polimi.it

***Abstract*—A service network with decentralized ownership is a system where nodes offering a variety of services are administered by different organizations – or even by a set of individuals. In such a context, nodes hosting services can dynamically enter and exit the system without prior notice, and there is no centralized point of control.**

If one wants to build into such a system the ability to direct incoming requests for the various hosted services to those nodes that can efficiently fulfill them, one option is to introduce in the system an entity that serves as a gateway to accept service requests, and is an intermediary to re-direct requests as needed. That implies that this intermediary is able to acquire and maintain accurate and up-to-date information on where it can direct incoming requests. Another option, which is the one we pursue in this paper, is to build the system as an overlay network, in which the nodes hosting instances of each of many different types of services can self-organize as "virtual clusters", and efficiently load-balance incoming requests amongst themselves. We describe our design and evaluation of a decentralized computing framework of this kind. We leverage a resilient peer-to-peer overlay that automatically re-configures its topology, responding to the number of different service types executing on the peer nodes, the dynamics of the participation of those nodes (peer churn), and the traffic coming into the system for the various services.

***Keywords*-load-balancing; self-organization; myconet; unstructured overlays; bio-inspired algorithms.**

I. Introduction

The provisioning of distributed computational services has followed a constant trend towards increasing degrees of virtualization. The original client/server model led to the evolution of dedicated clusters, followed by grids, and then by large-scale data centers able to host a multitude of diverse applications on virtual machines. That trajectory has now led to cloud computing, with its promise of opaque, elastic, and on-demand allocation of computational resources, and its computing-as-a-utility model.

Nowadays, cloud computing environments are typically quite large and complex, and under the centralized ownership and administration of a single entity, which provisions the resources, the networking and software infrastructure. However, there are also early signs of a further virtualization stage: a move away from centralized clouds, and towards *decentralized service networks*, in which the resources are not concentrated under a single ownership, but can be located anywhere on the network and administered by multiple organizations (or even individuals), and are engaged on the fly to satisfy clients' requests. This model borrows some traits from the idea of Edge Computing [1], [2], but expands it in a couple of new ways. First of all, it does not assume any hierarchical relationship between the various entities at the network's edge that put their resources together, in contrast with the classic Edge Computing model, which distinguishes between a primary origin location and one or more supplementary edge locations for resources; moreover, it is not limited to the planned deployment of a set of pre-defined services on well-chosen host nodes, but it embraces the SaaS (Software-as-a-Service) idea of the cloud, which strives to accommodate unanticipated and diverse client computing needs.

Decentralized service networks are currently being pioneered mainly by industry[1]. Research in this area is so far limited to a few, very recent, contributions (including [3], [4]), although this service provisioning model offers – side by side with its inherent promises – several interesting challenges. For example, the topology of a decentralized service network changes all the time, with participating nodes leaving and joining without notice or control, resulting in churn patterns that may be very different from what is expected in a regular cloud. Also, to build and operate a decentralized service network, one needs the ability to direct incoming requests to nodes that host services able to satisfy those requests, and, to effectively share the load among those nodes. To achieve that, one option is an entity that serves as a gateway or intermediary, accepting requests and re-directing them as needed to the appropriate nodes. That implies, however, that such an intermediary must maintain accurate and up-to-date information on where all services are located in the decentralized service network. That may become impractical as the scale of system increases.

We have investigated an alternative: an unstructured overlay network, in which the nodes that host instances of each of many different types of services self-organize into *virtual clusters*. Nodes in a virtual cluster do not need to be physically close, rather, they are logically interconnected. They must be able to efficiently load-balance incoming requests among

*All the activities for carrying out this work, including research and writing, have been equally divided among all the authors, who are listed in reverse alphabetical order.

[1]Examples include www.symform.com in the segment of storage services, and www.spotcloud.com, which harvests and brokers computational resources in a potentially global marketplace.

themselves, while the overlay as a whole is responsible for routing those requests to the right virtual cluster. We present here the design and evaluation of such a decentralized and self-organized service provisioning framework, focusing on the load-balancing problem outlined above. We have leveraged our previous experiences with load-balancing in heterogeneous and dynamic networks [5], and a bio-inspired superpeer-based overlay substrate that can automatically re-configure its topology, called Myconet [6]. Simulation-based evaluation shows how our prototype, which we code-named Mycoload, deals effectively with a number of dynamic dimensions, including the churn of nodes participating in the decentralized service network, the variation in the number of service types hosted by those nodes, the fluctuations of the traffic entering the network, and self-healing following highly disruptive network events.

The rest of the paper is organized as follows: in Section II the background works on which Mycoload it is based are presented, in Section III our proposed approach is described while the results obtained are shown in Section IV. Finally, related works are discussed in Section V and conclusions are drawn in Section VI.

II. Background

A. Self-organized service load-balancing

Figure 1 shows an example of a decentralized service network. Nodes, shown as circles, offer different services (characterized by different colors), and are organized in an overlay network highlighted by the arcs connecting the nodes. For an efficient execution of service requests, nodes must redirect the requests exceeding their processing capabilities to other nodes that are able to handle them. To achieve this goal in an efficient way, we need to guarantee that nodes have in their neighborhood a good number of other nodes offering the same kind of service. This property has to be guaranteed also in the presence of network churn, which can occur due either to failures in the system, or to nodes entering and leaving the network as part of their normal operation.

The approach presented in [5] addresses this problem through a load-balancing algorithm composed of two different reconfiguration algorithms that are executed in parallel.

The first algorithm performs clustering, continuously swapping links between neighbors in order to increase the links between nodes of the *same type*, i.e., nodes offering the same service. The rule that is applied is the following: a random node a at random intervals asks a random neighbor b for the reference of a node c not connected to a; if the type of a is equal to the type of c, or if the type of a is different from the type of b, then the existing link between a and b is replaced with a new link between a and c. This rule guarantees that the total number of links between nodes of the same type in the system is non-decreasing, and increases the probability that a more clustered situation is eventually reached.

The second algorithm is the dimension exchange algorithm [7] that focuses on balancing the load among neighbor nodes of the same type. A random node a at random intervals asks a random neighbor b of the same type about the current

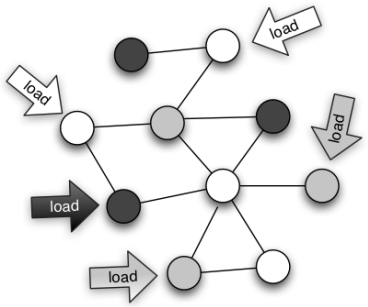

Fig. 1. Decentralized Service Network.

size of its service requests queue; if the queue size of b turns to be equal to the queue size of a, then no action is performed, otherwise one or more service requests are transferred from a node to the other in order to have the same amount of requests in the queue (with a possible slack of one).

In [5] we have shown that after a certain number of simultaneous execution of the two above algorithms, the throughput of the whole system, defined as the number of service requests successfully processed in a certain amount of time, approaches to the 90% of the theoretical optimum. Moreover, we have shown how that load balancing approach remains efficient in the presence of network churn.

While this approach appears to perform well in the cases we have studied, it presents some limitations. First of all, it assumes that nodes have homogeneous computational capabilities, and hence similar processing rates for incoming requests. That homogeneity cannot be guaranteed in a decentralized service network; on the contrary, the variability in computational power of the participating nodes may be quite significant. In that context, with an algorithm like the one presented above, a node with significantly above-average computational capabilities may not receive more requests than an average node of the same type. Another concern is that the overlay induced by the algorithm may be vulnerable to network disconnection. The objective of the algorithm in [5] is not towards constructing a redundant topology, nor does it incorporate specific provisions for resiliency to network disruptions; so in some cases, when random churn affects certain nodes, "islands" of same-type nodes can appear, which are not able to exchange requests with the rest of the network.

For these reasons, we have decided to adopt Myconet [6] for managing the links among the nodes in our decentralized service network. Myconet is a self-organizing protocol for creating and maintaining superpeer overlay topologies in unstructured peer-to-peer networks. We aim at leveraging its superpeer topology to explicitly account for (and exploit) the heterogeneity of nodes, and arrange same-type clusters around the nodes with the most computational capabilities. That can result in performance benefits, including increased convergence rate (i.e., the amount of time that is needed to obtain a satisfactory configuration) and higher throughput. Myconet overlays have also demonstrated nice self-repairing properties, and can recover quickly from even the most pronounced network-disrupting events.

B. Bio-inspired superpeer overlay

The design of Myconet takes inspiration from the growth patterns of the root-like, filamentous structures used by many species of fungus for collection of nutrients and reproduction. The entire root network of fungus is referred to as a *mycelium* and an individual strand as a *hypha*. Natural hyphae possess many properties that are also desirable for distributed systems, including an efficient balance of exploration and exploitation of resources, and robust adaptation in the face of damage or environmental changes.

Each peer in Myconet is characterized by its "capacity", an abstraction used to represent a peer's ability to service the needs of other peers. That abstraction can be cast to different characteristics, which may be specific to the application running on the Myconet overlay. For example, in the current work, we have mapped peer capacity to the rate at which a node in the service network can process incoming requests. The capacity is not necessarily in relation with the actual hardware resources of the node hosting the service as the processing rate may depend on many factors (e.g. external load on the node, software version, operating system, etc)[2].

Myconet uses a set of local rules and a hierarchy of peer states that are loosely modeled on hyphal growth dynamics. Peers may either be *hyphal* peers (i.e., superpeers that provide additional services to other peers) or *biomass* peers (less-powerful peers that rely on hyphal peers and their services when available). Hyphal peers switch between three further hierarchical protocol states: *extending*, *branching*, and *immobile*. Each of the states above has a different role in managing the topology of the overlay.

All peers begin as biomass, and follow a multi-step promotion/demotion process to and from the various superpeer states. *Biomass* peers perform only minimal roles in management of the topology. *Extending* hyphae act as attachment points, and continuously explore the network looking for new or isolated biomass peers to connect to. *Branching* hyphae grow new cross-connections with other hyphal nodes; they also regulate the number of extending hyphae in the network, at times selecting large-capacity biomass peers for promotion to the extending state, and at other times targeting excess extending hyphae for demotion. *Immobile* hyphae similarly control the number of branching peers in the network, and ensure the chosen level of cross-connectivity between hyphal peers remains stable; immobile peers are selected by the protocol dynamics as being of the highest capacity (and, implicitly, reliability, as those peers have remained in the network long enough to be promoted to this state).

The topology management rules followed by a Myconet peer are determined by its protocol state, as shown in Figure 2. For details on all the rules and state transition criteria, we direct the reader to our previous work [6]. In brief, Myconet has a set of promotion/demotion rules, which self-regulate

[2]In this project, each peer in a Myconet overlay is also a node in the decentralized service network; therefore in the remainder of this paper, we will use the terms *peer* and *node* interchangeably.

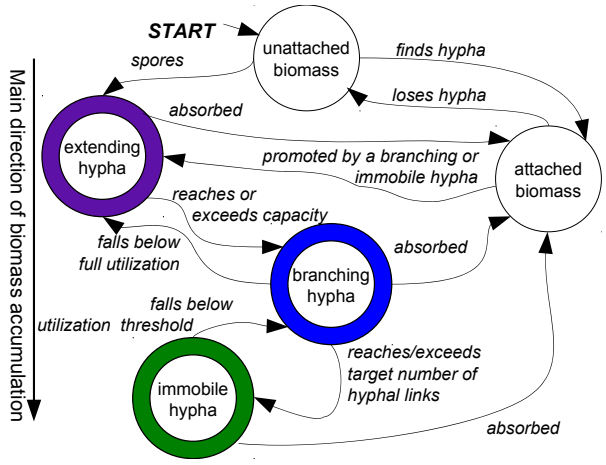

Fig. 2. Myconet protocol state transitions. Peers adaptively transition between regular peers (biomass) and three varieties of superpeers (hyphae) depending on changing conditions.

the superpeer population in comparison with the amount and capacity of biomass peers in the network. Promotion and demotion are predicated on a few parameters: a target number of connections each hyphal peer should have to biomass peers (B_S), which is proportional to the capacity of that hyphal peer; and a target number of *hyphal links* it should "grow" towards other superpeers (C_S), to ensure the robustness of the overlay. C_S is a customizable parameter of the protocol itself.

Myconet also has rules that regulate the interactions between neighboring hyphal peers; these rules, on the one hand, aim at transferring biomass peers towards the higher-capacity hypal peers that can best service them, while, on the other hand, ensure that hyphal peers that cannot efficiently reach their B_S or C_S targets will become candidate for demotion.

It is important to notice that all Myconet rules operate locally. Peers make decisions based on their individual neighborhood, that is, the set of peers to which they maintain links. These links exchange information used for topology adjustment, maintain strong interconnections to increase robustness in case of superpeer failure, and act as the communication substrate for application-level protocols running on the overlay. For several protocol operations (bootstrapping and growing new inter-hyphal connections) random non-neighbor nodes are selected, through the use of a gossip-based mechanism that maintains a cache of known peers and the protocol states of those peers. For our current implementation, we use a simplified version of the Newscast protocol [8].

As a result of the design of these protocol rules, Myconet overlays are self-healing in the case of peer failure, ranging from normal churn to malicious attacks directly targeting the superpeers. The network quickly re-organizes and repairs itself by growing new hyphal links and promoting new peers as needed, and has been demonstrated to be particularly effective and fast in recovering an efficient network configuration, even under especially disruptive events.

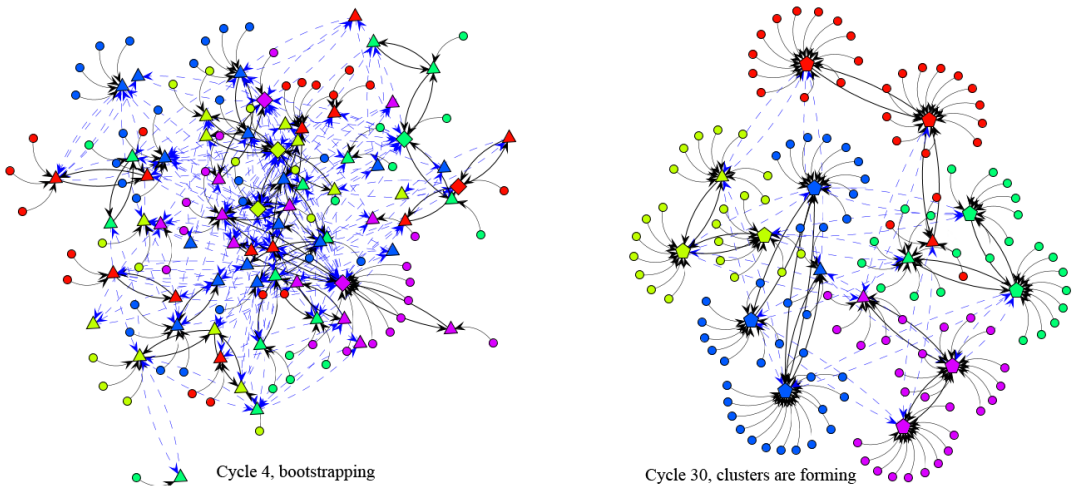

Fig. 3. Mycoload clustering.

III. Mycoload: biologically-inspired clustering and load-balancing

We have extended Myconet in order to make it suitable to serve as the overlay management system for our load balancing approach. The load-balancing algorithm has also been modified to take advantage of the new overlay structure. We augmented the basic Myconet protocol with rules that support the clustering of same-type peers; on top of that efficient topology, self-organized load-balancing is applied to the queued requests for the various service types residing in a decentralized service network.

Myconet aims at selecting the highest-capacity peers to serve as superpeers. The first application-specific design decision we made for our load-balancing scenario was therefore to map the capacity of a peer to the amount of service requests that peer is able to fulfill in a time unit. This, in turn, is reflected in the Myconet overlay as the target number B_S of client peers maintained by a superpeer, as we want higher-capacity peers to connect to a large number of lower-capacity same-type neighbors. This way, superpeers are well positioned to efficiently balance their neighbors' request queues.

Moreover, as Myconet had originally been developed to manage peers irrespective of service type, another extension to the protocol was to make it aware of peer types. A Mycoload peer may have at any given time both same-type and different-type neighbors, but, as discussed in section II, a peer can only perform load-balancing operations with same-type peers. Therefore different types of rules apply when that peer manages its links to same-type vs. different-type neighbors, to ensure the incremental construction of neighborhoods in the overlay, in which one or more hyphal peers aggregate and serve a number of biomass peers of the same type.

The current implementation assumes that peers provide only one service type, and hence participate in only one cluster. Under this model, peers could offer more than one service type by running multiple instances of the protocol. Here we present results for single-type peers only; extending Mycoload to allow peer to offer multiple service types is planned as future work.

The customization and extension of protocol rules used in the Mycoload application for same-type and different-type neighbors is discussed in Section III-A. In Section III-B we discuss how the load-balancing algorithm has been customized to leverage the superpeer topology with same-type clustering induced by those rules.

A. Cluster Construction Rules

To enable the clustering of same-type peers, Mycoload uses a set of rules that regulate the interactions of same-type vs. different-type peers. We focus here on those rules, and mostly gloss over the mechanics of superpeer selection, promotion and demotion, which remains the same as in the Myconet basic protocol.

Biomass Peers: All peers begin protocol execution as biomass. A disconnected biomass peer b will attempt to connect to an extending superpeer it can find through the lower-level gossip protocol. However, it will only select one of its same type. If no suitable extending superpeer can be located, b will promote itself to extending status. This rule ensures that there are always extending hyphae of each type in the network, since these self-selected extending peers must act as aggregation points for isolated peers of their type. If a biomass peer ever becomes disconnected, it searches for a new extending hypha to connect to, as during bootstrapping.

Hyphal Peers - all states: Hyphal peers aggregate biomass peers of their same type up to a target level B_S (proportional to their capacity); to that end, hyphal peers in a higher protocol state will always try to "pull" biomass from other neighboring hyphal peers of their type in lower states, or in the same state but of lower capacity, through the execution of *absorption* rules.

Hyphal peers also form the backbone of the overlay, and are responsible for its robustness; to that end, hyphal peers attempt to create and maintain a target number of links C_S to other hyphae of their type, which is a parameter of the protocol. In Mycoload, there is another parameter, as hyphal peers also try

to maintain C_O links to hyphae of *different types*. C_O ensures that clusters do not become disconnected from one another.

Increasing the C_S or C_O parameters increases the number of cross-connections (and the size of each peer's neighborhood views), while adding overhead to the protocol. C_O ensures that clusters do not become disconnected from one another.

Hyphal peers act as type matchmakers for other peers; if a hypha h has a neighbor n of a different type, h will check if some other of its neighbors s is of the same type as n. If so, h will transfer n to become a neighbor of s. If not, h tries the same thing with the neighbors of all of its own neighbors. h will not necessarily drop its link to n, if it remains within the limit of its target C_O different-type hyphal links.

Hyphal peers also execute rules specific to their protocol state, as described below.

Extending Peers: Superpeers remain in the extending state until they have reached their target number of biomass neighbors of the same type B_S, at which point they promote to branching status. Extending peers also anchor themselves to the overlay with a single link to either a branching or an immobile peer of any type. In the event two extending peers of the same type become neighbors, the lower-capacity peer will transfer all of its biomass peers to the higher-capacity and demote itself to biomass status. This is an absorption rule.

Branching Peers: One primary function of these superpeers is to grow new links to other superpeers, thus building robust cross-connections for the overlay. Branching peers are those that have aggregated their target number of same-type biomass peer (B_S), but have not yet reached their targets of C_S and C_O hyphal links to other superpeers of the same type and different types, respectively.

With respect to same-type hyphal links, if a branching peer h_b is under the target number C_S, it will search its neighborhood to determine if any of its neighbors are attached to a suitable same-type hyphal peer that is not already its own neighbor. With respect to different-type hyphal links, if h_b is under its target C_O, it will randomly select a peer from the gossip cache and grow a neighbor link to it. The resulting peer might be of any type (including the same). A branching peer also attempts to always have one extending peer among its neighbors; if no such peer exists, it will pick its largest capacity biomass child and promote it, obtaining an extending peer of its same type.

If a branching peer ever gets over its biomass capacity, it will push excess biomass children to an attached same-type superpeer. Once it has grown C_S and C_O hyphal links, a branching peer will promote to immobile state.

Immobile Peers: Once a peer has achieved immobile status, it will attempt to maintain it through absorption, that is, by pulling biomass from lower-state hyphae of its same type (possibly resulting in their demotion). Similarly, it will try to grow new hyphal links if some are lost, in order to maintain the target C_S and C_O. If the immobile peer, instead, happens to have more than C_S hyphal links to same-type superpeers, or or more than C_O hyphal links to different-type superpeers, it will randomly drop the excess links.

An immobile peer should be connected to either zero or one same-type extending peers. In case it has multiple extending neighbors hyphae, it will connect them together, thus triggering the absorption rule. If it has both branching and extending neighbors, it will transfer the extending peers to become children of the branching peers. In order to prevent the overlay from settling into a local optimum, immobile peers will occasionally grow a new random link by picking a random hypha from the gossip cache and connecting to it.

The addition of the rules described above to the basic Myconet protocol enable Mycoload to quickly settle into an overlay configuration where peers of each type are concentrated around, and connected to, one or more hyphal peer of that type. Figure 3 shows with color codes how such a self-organizing clustering occurs. The first snapshot shows the overlay during bootstrapping phase (at round number 4, when small clusters are beginning to self-aggregate), and the second snapshot shows the overlay after it has stabilized its topology at round 14. These figures were taken from a Mycoload run with 150 peers, 5 types, $C_S = 2$, $C_O = 2$, and max peer capacity of 15.

B. Load-Balancing

Load-balancing is – as before – performed between neighbors. However, in the previous work, since all peers were considered homogeneous in their computing capabilities, the goal of the algorithm was to balance the queues of service requests for neighboring peers as uniformly as possible. In Mycoload, because of the heterogenous capacity of peers, the goal of load-balancing becomes making queue lengths proportional to the capacity of each peer.

Based on this principle, when a load-balancing operation is performed, a peer p_a selects a random same-type hyphal neighbor p_b. If p_a is a biomass peer, it will have only a single neighbor, a same-type hypha. If p_a is a hyphal peer, it will only try to balance its queue with other same-type hyphal peers; as biomass peers have only a single neighbor, they will be the ones to initiate the load-balancing operations with their parent. This way, load-balancing operations tend to occur preferentially between the more capable superpeers, which helps ensuring that jobs will be balanced throughout the cluster. p_a and p_b compare their queue lengths; "Ideal" queue lengths are calculated by determining the total number of jobs in both queues and dividing the jobs proportionally based on p_a and p_b's capacities. If the queues are unbalanced, jobs are transferred from the queue that is over its ideal length to the queue that is under that length.

During load-balancing operations, the transfer of a job involves only a transfer of a reference to the job, that is, to the location where that job can be actually retrieved when a peer is ready to execute it. The choice of forwarding only references helps to keep the network overhead as small as possible, since jobs could be composed of large amounts of data.

TABLE I
MYCOLOAD PROTOCOL PARAMETERS

Symbol	Description	Value
C_s	Target number of peers of the same type	5
C_o	Target number of peers of different type	3
MN_{period}	Activation period Myconet protocol	1
LB_{period}	Activation period load-balancing protocol	1

IV. EXPERIMENTS

In this section we give a detailed description of the experiments developed to evaluate the proposed technique. We have chosen a simulation approach because the size of the decentralized service networks we consider is not suitable for the application of formal methods, usually exploited for analyzing systems with a small number of peers. Also, the peers can evolve non-deterministically in a large number of different ways. We discuss how the experiments have been set up, the choice of parameters and the metrics we extracted. Finally, we describe the scenarios chosen to evaluate Mycoload and the results we have obtained.

A. Simulator

The load-balancing technique proposed in this paper has been evaluated using simulations implemented on top of the Java-based PeerSim platform [9]. PeerSim can be used according to two simulation models: *cycle-based*, in which peers get the control after a fixed constant time step in a sequential fashion, and *event-based* in which the cycle-based assumption is removed and simulator components can be active at any time, exploiting full concurrency. We decided to adopt the cycle-based model in order to ensure our experiment could scale easily to large number of peers, at the cost of little loss of precision in the results.

In order to reduce the variability of results, we performed 20 independent runs for each experimental configuration, for which mean and standard deviation have been computed.

B. Parameters of the simulation

The parameter space of systems involving thousand of peers can be very large, and careful selection is of utmost importance in order to create representative experiments for the most common scenarios. Table I lists some parameters that are specific to the Mycoload protocol and that have been fixed after some experimental trials and sensitivity analyses, which are not reported here *in toto* due to space limitations.

For the experiments discussed in this paper we have used values of $C_S = 5$ for the target number of same-type hyphal links (the Myconet default) and $C_O = 3$ for the target number of different-type hyphal links, although we also tested the protocol with values of $2 \leq C_S \leq 6$ and $2 \leq C_O \leq 6$. The sensitivity of the network to values of C_S and C_O lower than the default is limited: the time to convergence and sensitivity to churn of the network increase somewhat, but the performance of the service network in processing requests remains similar. Higher values did not result in improved performance, but showed increased messaging overhead.

Other parameters used in the simulations depend on the specific scenario to be tested. In particular, we vary the network size composing the system from 100 to 5000 peers. The number of distinct services in the network is important, as it puts under pressure the ability of Mycoload to form many interconnected clusters of same-type peers. In our experiments we vary the number of service types up to 100. A peer's capacity (an abstraction representing the number of service requests it can process within a single simulation cycle) is set according to a power law distribution: the probability of a given peer n to have a capacity c_n is $P[c_n = x] = x^{-\alpha}$ where $1 \leq x \leq c_{max}$. In the experiments we fixed $\alpha = 2$ and $c_{max} = 10$ (which assumes that the most powerful peers are no more than an order of magnitude more powerful than the least powerful peers). The number of simulation rounds is fixed to a maximum of 600, as all experiments converge to a stable state before this limit.

Service requests are generated in bursts that are periodically injected into the system. The period between bursts is a simulation parameter that we call *Service Request Period* (SRP). The size of each burst (BS) is calculated according to the following formula: $BS = SRP \times C \times SLP$, where $C = \sum_{1 \leq n \leq N} c_n$ represents the total processing capacity of a system containing N peers, and SLP (the *System Load Percentage*) represents the percentage of each SRP that an ideal system composed of a single peer of capacity C would use to process all requests belonging to the burst. In order to experiment with different levels of load for the system, SLP is another simulation parameter. As an example, if we consider $SRP = 10\ cycles$, $C = 100\ req/cycle$ and $SLP = 50\%$, the size of a burst will be of 500 requests. Once generated, service requests are forwarded toward random peers of the corresponding type.

C. Setting up the experiments

In order to correctly evaluate the effectiveness of the approach, some considerations need to be made regarding the experimental metrics to observe and the experimental scenarios associated with them.

The identified metrics are the following: (i) *Response Time Optimality* (RTO), defined as $RTO = \frac{ORT}{MRT}$ where ORT is the Optimal Response Time of an ideal system composed of a single C capacity peer and MRT is the Measured Response Time. RTO gives an idea of how much the measured response time is far from being equal to the optimal one of the ideal system. (ii) The number of *exchanged messages*, which is important to understand the network costs for both the tasks of peer clustering and load-balancing; (iii) the *converged optimality*, which represents the RTO calculated once the system has converged to a stable state.

The previous metrics allow us to evaluate the following properties of Mycoload: quality of achieved result, speed in reaching a stable state of the network, and robustness of the overlay. We decided to test Mycoload against three completely different scenarios which are representative of a wide range of dynamics for a decentralized service network.

- *Stable*. In this scenario the network is static, in the sense that no peers join or leave the existing network for the

course of the experiment.

- *Churn*. In this scenario the network is subject to the continuous dynamic phenomena typical of peer-to-peer networks. Specifically, we purposely remove a percentage $churn_{\%}$ of peers from the network for each service and immediately add again the same percentage of peers. New peers enter the network in a disconnected state.
- *Disruptive*. This scenario replicates the network dynamics happening during a catastrophic event (i.e. network partitioning, malicious attacks, etc.). Specifically, a large portion of the network (ranging from 30 to 80%) is removed and then added again, thus testing Mycoload's ability to heal the network after a major disruption.

D. Stable Scenario Experiments

In this subsection we analyze the experimental results obtained in the case of the stable scenario. The primary benefit we hypothesize is a significant increase of the RTO.

Reference Experiment: In Figure 4 we can see a curve representing a typical evolution of RTO (referred as optimality in the *y*-axis) with respect to simulation cycles (*x*-axis) and the cost needed to obtain it in terms of exchanged messages (the mean of 20 simulation runs is plotted in red with standard deviation indicated using a lighter-color band). This reference experiment has been run with 1000 peers, with $SLP = 70\%$, 100 different service types, and no churn. In all these experiments the load-balancing algorithm begins to execute at cycle 50 to highlight its effects with respect to the situation without Mycoload (in the first 49 cycles). We can see that the maximum RTO of 94% is obtained in 83 cycles after the algorithm is started (i.e., at cycle 133) and can then be considered stable, meaning that the algorithm is able to efficiently balance the load.

The cost for increasing the RTO has been measured in terms of message overhead added by topology reconfigurations and load-balancing operations. We have measured a constant average message overhead of about 3.0 messages per cycle per peer for the load-balancing operations, and 4.7 messages per cycle per peer for the topology reconfiguration operations. Both load-balancing and topology reconfiguration messages can be assumed to be small in size since load-balancing messages contain only references to the actual requests (thus they do not depend on the size of the request), while topology messages contain only basic topology modifications operations. In the next paragraphs we will show how the RTO and the messages are affected by changes in the simulation parameters.

Impact of variations in the number of service types: In these experiments we have used the same parameters of the reference experiment, but we have varied the total number of service types. The charts we have obtained have very similar dynamics over time as the ones shown in Figure 4, but with some differences reported in rows 3-6 of Table II. The algorithm convergence is slightly faster as the number of service types decreases, while the RTO obtained at convergence is not significantly affected with respect to the reference experiment (bold line of Table II). As the number of service types gets higher, the total number of messages required by the load-balancing algorithm does not change appreciably, while the number of topology reconfiguration messages generally tends to increase. The explanation for this is the fact that a large number of service types makes the clustering part of the topology reconfiguration more difficult, which increases the message traffic in that category.

TABLE II
EXPERIMENTS VARYING THE NUMBER OF PEERS, SERVICE TYPES, AND SLP. RTO AT CONVERGENCE AND CYCLES TO REACH CONVERGENCE ARE SHOWN ALONG WITH MESSAGE OVERHEAD.

#	Peers	Types	SLP	Cycl.	RTO	Top. msgs	LB msgs
1	100	10	70%	76	96%	2.75	1.60
2	100	10	90%	138	89%	4.63	3.35
3	1000	20	70%	75	91%	4.09	3.03
4	1000	40	70%	72	92%	4.24	3.01
5	1000	60	70%	74	93%	4.38	3.01
6	1000	80	70%	79	94%	4.54	3.01
7	1000	100	30%	64	77%	4.70	2.24
8	1000	100	50%	64	79%	4.70	2.62
9	**1000**	**100**	**70%**	**83**	**94%**	**4.70**	**3.02**
10	1000	100	90%	222	92%	4.70	3.46
11	5000	100	70%	82	91%	4.09	3.03

Impact of SLP variations: These variations study how the results of the reference experiment are affected when generating different percentages of load in the system (SLP). We still obtain charts with the same shape as the reference experiment, but some quantitative differences, which are shown in rows 7-10 of Table II. RTO at convergence, in particular, becomes lower for underloaded networks (77% RTO given 30% SLP), while it is still quite good with networks with high load (92% RTO given 90% SLP). The explanation for this result in underloaded networks is due to the fact that many peers of the network eventually have an empty queue, therefore they frequently become idle before receiving new requests from other peers with non-empty queue. The fact that the network has increasing queues explains the lower optimality obtained in a overloaded network, since even at convergence, it processes fewer requests than the ones it receives.

Impact of variations in the number of peers: This experiment variation regards the scalability of the approach. Figure 5 shows the scenario of a network of 5000 peers, 100 types, and $SLP = 70\%$. As we can see from the figure and Table II, performance is very similar to the reference experiment. The differences are due to the different proportion between the total number of peers (now 5000) and the number of service types (still 100). In this scenario, Mycoload converges quickly to around 91% RTO.

E. Experiments in scenarios with network dynamism

In this subsection we present results for configurations equivalent to the previously discussed stable scenarios with the addition of churn, as well as more pronounced disruptions. These experiments are important since we expect self-organization algorithms like the one we propose to adapt to unpredictable patterns of peer entry and exit, and to self-heal in the face of major damage to the network.

Effects of continuous churn: Churn is introduced at the start of the simulation and continues throughout its execution.

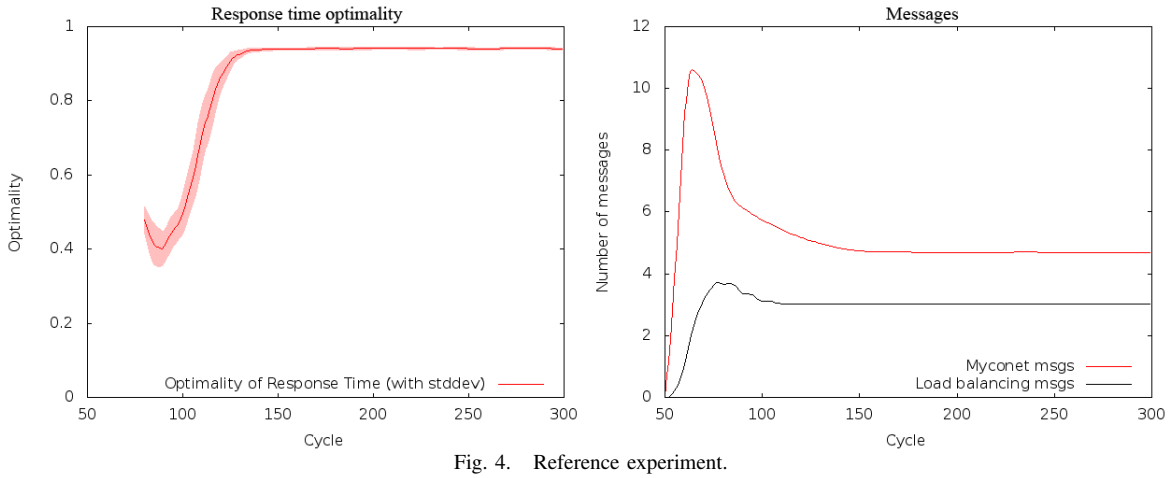

Fig. 4. Reference experiment.

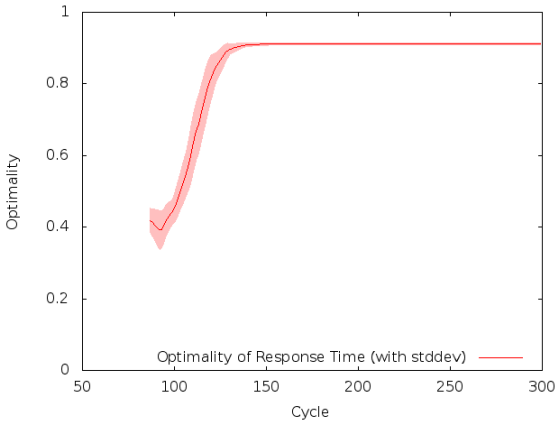

Fig. 5. Experiment variation with 5000 peers.

The results are shown in Figure 6. Churn is represented by removing a portion of the peers every ten simulation cycles, and immediately re-introducing the same number of peers. In the four churn scenarios, an average of 5%, 10%, 15%, and 20% of the total number of peers in the network are removed, with the exact number selected from a Poisson distribution with the appropriate average.

If we consider one cycle as representing a time interval in the order of seconds in the real-time domain (which is certainly adequate to execute the algorithms discussed so far), these levels of churn would represent quite significant rates of peer turnover for a service network. In those conditions, the overlay responds smoothly to increasing levels of churn; 5% churn results in a converged optimality of around 0.90, whereas optimality is around 0.78 at 20% churn (Figure 6a).

For all scenarios, the average number of load-balancing messages per peer remains stable at around 2.6, while an increasing number of messages must be sent to adapt the overlay based on the changing network conditions; each 5% increase in churn results in about 1 additional message per peer each cycle. Whereas a stable network has a topology management overhead of around 4.7 messages per peer, 5% requires 5.8 messages per peer, 10% requires 6.8, 15% requires 7.6, and 20% incurs 8.3.

Effects of major network disruption: Figure 7 shows what happens to the RTO when a pronounced disruptive event occurs. We used the same configuration as the reference experiment, and the disruptive scenarios we considered replace 30%, 50%, and 80% of peers at cycle 300 (after RTO stability has been reached). We observe in all cases a rapid self-healing behavior after the disruption. Also in this case a cost is paid in the form of topology reconfiguration messages; killing 30% of the peers causes a spike of 8.1 messages per peer per cycle, while 50% has a spike of 9.4. The costs paid following an event where 80% (an extremely large proportion) of all peers are killed causes a spike of 10.8 topology messages per peer per cycle, which is comparable to the message spike that is visible during the bootstrapping phase of the protocol.

F. Discussion

The experiments described above show the benefits and the limits of our approach. In particular, Mycoload's strengths are the reduced number of messages and increased convergence rate with respect to our initial work [5]. In that work a maximum network throughput (processed jobs per time unit) of 90% of the optimal one was reached after more than 1000 messages per peer (with a SLP of 100%). We ran the same experiment with Mycoload and found that the same throughput percentage was reached after just 145 messages per peer, and that optimal throughput (100%) was achieved after 529 messages per peer. We attribute this improvement to the more rational structure of the network topology that considers key properties of the peers, such as their capacity. The drawbacks of Mycoload are its slower convergence when the churn is very large, and a reduced ability to optimize the RTO when the network is subject to only a light load.

V. Related Work

The literature concerning the problem of load-balancing is quite extensive and includes solutions for a large variety of scenarios, ranging from multiprocessor computers, to telecommunication networks, content delivery networks, grid environments, and more.

Mycoload's approach is related to other topology construction and management protocols for unstructured peer-to-peer networks such as T-MAN [10] and SG-1 [11]. T-MAN uses a ranking function to determine relative distances of a peer to other peers in its local neighborhood. This approach enables

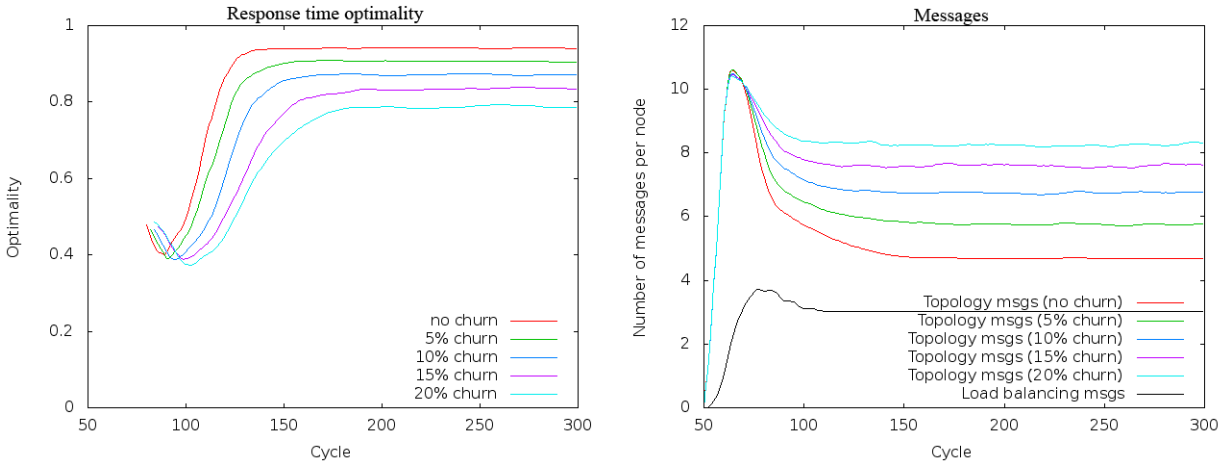

Fig. 6. Experiment variation with varying churn levels.

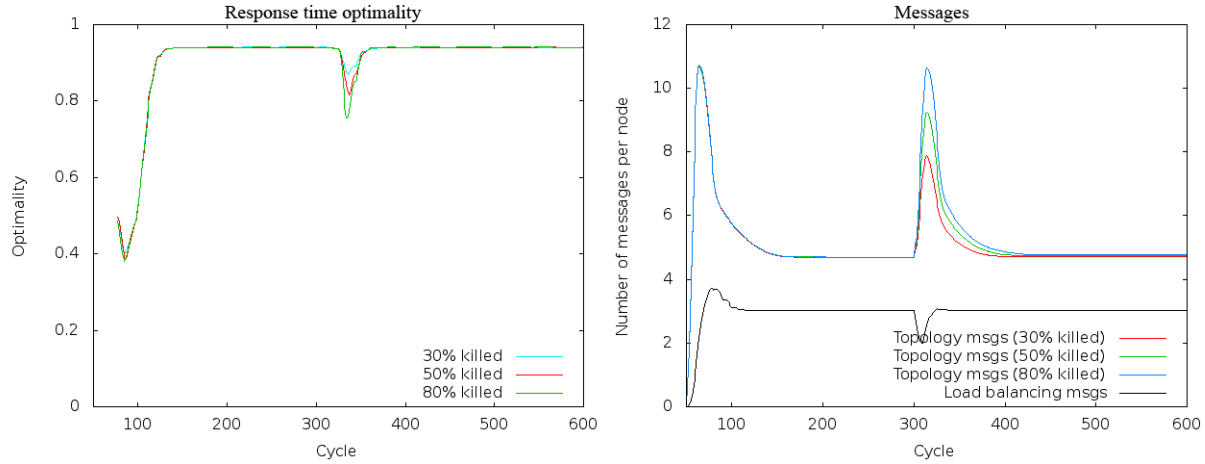

Fig. 7. Experiment variations with disruptive event at cycle 300.

clustering but does not address load-balancing efficiency. Mycoload improves load balanching by using intra-cluster hierarchies to position more capable nodes centrally within each cluster. Due to space limitations, readers are referred to [6] for a more extensive discussion of topology management in the Myconet protocol.

Recently the problem of load-balancing in decentralized service networks has gained attention: in [4] Ranjan et al. propose a new software fabric to balance service provisioning requests among a set of VMs deployed in a cloud environment. This approach differs from ours as it relies on a structured peer-to-peer network model (i.e., a distributed hash table). Other works such as [12], provide some guarantees on the convergence time of the network to a stable state. However, they only deal with a fixed initial assignment of requests to each peer and no network dynamism concern is taken into account (i.e., the topology is stable).

Lately, the increasing diffusion of ubiquitous and pervasive frameworks living in extremely dynamic environments, has escalated the need for robust and adaptive load-balancing techniques. In [13] the authors achieve a load-balancing behavior by modifying the degree of each node proportionally to the available computational resources at the node itself. The execution of a job within a node causes one of its links to be removed while its completion adds a new link toward a new node. Biased random walks are used to find the best node that is going to execute the job. The authors obtain performance that seems similar to our previous work [5], but they do not consider the case of nodes with different processing capabilities and the effects of churn.

Another approach has been proposed by Li and Kameda [14]. They consider a system with heterogeneous nodes and jobs, meaning that each node is able to process all the jobs, but with a different rate for each job type. In this case the load-balancing problem is described as an optimization problem decomposed into simple rules to be run at node level in a decentralized way. The difference with respect to our approach is that their work does not address the possibility of changes to the topology and does not consider issues of churn and scalability.

The problem of load-balancing has also been considered in other domains. Examples are structured peer-to-peer networks to manage distributed hash tables (DHT). In this context, the objective is to balance the association of keys to nodes according to nodes capacity. Also in this case, random probing of candidate nodes for load-balancing is used as a robust technique against churn [15], [16]. In telecommunication networks problems of routing and load balancing have been addressed

by [17], [18] borrowing concepts from biology. In particular, authors exploit the emergent behavior arising from the interaction of virtual "ants" with the surrounding environment. Ants move within the environment and leave information on it (i.e., pheromones) which are used by other ants to modify their behavior. This technique presents characteristics similar to the ones we have been able to achieve in our work (i.e., resilience to network fluctuations, decentralized control, emergent behavior, etc.). In the area of game theory, Penmsata and Chronopoulos [19]) solve dynamic load-balancing problems by combining existing static approaches. In this approach the nodes periodically communicate information about their state (i.e., capacity and load) and use this information to compute the right actions to reach a Nash equilibrium that would respect a situation of good load distribution. This approach may optimize global system metrics such as the system response time, but it does not consider different types of jobs, and the network scale is lower than the one we consider in this paper.

Finally, another class of techniques tries to address load estimation, a problem complementary to the one discussed in this paper. In [20], load estimation at each node shows the advantage of performing the load-balancing actions on each job arrival. Possible bandwidth limitations are considered together with heterogeneous nodes. In [21] each peer is able to run different services, and optimization is performed by randomly choosing an action from a set of run-time generated actions, with a probability proportional to its profitability and the evaluated positive effects on system load-balancing.

VI. Conclusions

In this paper we have discussed a self-organizing approach for balancing the load in networks composed of peers offering different types of services. The proposed approach combines and exploits the synergies between two existing self-organizing techniques: one for clustering peers with respect to the types of their services, and the other for creating and maintaining superpeer topologies. The implementation of the approach using the PeerSim simulator has shown that the proposed technique is able to obtain a significant reduction of the service response time when compared to our previous work under several system configurations and environmental conditions. We have also seen that the approach also inherits the typical benefits of bio-inspired self-organization, such as the scalability with respect to the number of peers, and the resilience to dynamism and unexpected system behavior.

In the future, this work will be extended in the following directions: first, we will integrate the technique into an existing distributed cloud system to evaluate performance in a real system. Second, we will extend the approach to optimize additional parameters, such as the network utilization and the economic cost for using certain peers (assuming that peers can offer the same services with different costs). Third, we will extend the protocol to accommodate peers offering multiple service types. Last, but not least, in future the topology may be built also taking into account the locations of the different peers, which can be useful in multi-cloud systems.

Acknowledgment

This research has been partially funded by the European Commission, under project SMSCom (IDEAS-ERC 227977) and S-Cube (FP7/2007-2013 215483). Special thanks to Rachel Greenstadt for her contributions to the development of Myconet.

References

[1] A. Davis, J. Parikh, and W. E. Weihl, "Edgecomputing: extending enterprise applications to the edge of the internet," in *WWW Alt. '04*. New York, NY, USA: ACM, 2004, pp. 180–187.

[2] M. Desertot, C. Escoffier, P. Lalanda, and D. Donsez, "Autonomic management of edge servers," in *Self-Organizing Systems*, ser. LNCS. Springer Berlin / Heidelberg, 2006, vol. 4124, pp. 216–229.

[3] V. D. Cunsolo, S. Distefano, and A. Puliafito, "Cloud@home on top of reservoir," in *Cloud Computing*, ser. LNICST. Springer Berlin Heidelberg, 2010, vol. 34, pp. 41–56, 10.1007/978-3-642-12636-9_3.

[4] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," in *Cloud Computing*, ser. Computer Communications and Networks. Springer London, 2010, vol. 0, pp. 195–217.

[5] E. Di Nitto, D. J. Dubois, R. Mirandola, F. Saffre, and R. Tateson, "Applying self-aggregation to load balancing: experimental results," in *BIONETICS'08*. ICST, 2008, pp. 14:1–14:8.

[6] P. Snyder, R. Greenstadt, and G. Valetto, "Myconet: A fungi-inspired model for superpeer-based peer-to-peer overlay topologies," in *SASO'09*, 2009, pp. 40–50.

[7] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *J. Par. Distrib. Comput.*, vol. 7, no. 2, pp. 279–301, 1989.

[8] M. Jelasity, W. Kowalczyk, and M. Van Steen, "Newscast computing," Vrije Universiteit Amsterdam Department of Computer Science Internal Report IR-CS-006, 2003.

[9] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," `http://peersim.sf.net`.

[10] M. Jelasity and O. Babaoglu, "T-Man: Fast gossip-based construction of large-scale overlay topologies," *University of Bologna, UBLCS-2004-7, Italy*, 2004.

[11] A. Montresor, "A robust protocol for building superpeer overlay topologies," in *P2P'04*. Zurich, Switzerland: IEEE, Aug. 2004, pp. 202–209.

[12] M. Franceschelli, A. Giua, and C. Seatzu, "Load balancing over heterogeneous networks with gossip-based algorithms," in *ACC'09*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1987–1993.

[13] M. Randles, O. Abu-Rahmeh, P. Johnson, and A. Taleb-Bendiab, "Biased random walks on resource network graphs for load balancing," *J. of Supercomputing*, vol. 53, pp. 138–162, 2010.

[14] L. Jie and H. Kameda, "Load balancing problems for multiclass jobs in distributed/parallel computer systems," *IEEE Trans. Computers*, vol. 47, no. 3, pp. 322–332, March 1998.

[15] H. Shen and C.-Z. Xu, "Locality-aware and churn-resilient load-balancing algorithms in structured peer-to-peer networks," *IEEE Trans. Par. and Distr. Syst.*, vol. 18, no. 6, pp. 849–862, June 2007.

[16] S. Fu, C.-Z. Xu, and H. Shen, "Random choices for churn resilient load balancing in peer-to-peer networks," in *IPDPS*, 2008, pp. 1–12.

[17] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," *IEEE Trans. Systems, Man and Cybernetics, Part A*, vol. 33, no. 5, pp. 560–572, Sept. 2003.

[18] R. Schoonderwoerd, O. Holland, and J. Bruten, "Ant-like agents for load balancing in telecommunications networks," in *AGENTS '97*. New York, NY, USA: ACM, 1997, pp. 209–216.

[19] S. Penmatsa and A. Chronopoulos, "Dynamic multi-user load balancing in distributed systems," in *IPDPS'07*, 2007, pp. 1–10.

[20] R. Shah, B. Veeravalli, and M. Misra, "On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 12, pp. 1675–1686, 2007.

[21] N. M. Calcavecchia, D. Ardagna, and E. Nitto, "The emergence of load balancing in distributed systems: the selflet approach," in *Runtime Models for Self-managing Syst. and Appl.*, ser. Autonomic Systems, D. Ardagna and L. Zhang, Eds. Springer, 2010, pp. 97–124.