An Adaptive Clustering Approach for the Management of Dynamic Systems

Carmelo Ragusa, Antonio Liotta, *Member, IEEE*, and George Pavlou, *Member, IEEE*

***Abstract*—Adaptive clustering is one of the fundamental problems behind autonomic systems and, more generally, an open research issue in the area of networking and distributed systems. The problem of giving structure to large-scale, dynamic systems through clustering and of electing centrally located nodes (cluster heads) is nontrivial. This is in fact an NP-complete problem when striving for optimality. We propose an innovative strategy based on code mobility that dynamically computes near-optimal clusters in linear time. Our approach is autonomic, does not require any user intervention, is self-configuring, self-optimal, and self-healing. We demonstrate these features through an extensive set of simulations, discussing the viability of the algorithm based on state-of-the art technologies, and elaborating on its applicability to distributed monitoring, peer-to-peer systems, application-level multicast, and content adaptation networks.**

***Index Terms*—Autonomic communication systems, clustering methods, code mobility, network partitioning, self-healing, self-management.**

I. Introduction

THE TREMENDOUS developments surrounding Internet-based systems are making ubiquitous, pervasive communications a reality. In this context, mobility is a key element associated with ubiquitous access to multimedia services and applications. On the other hand, new distributed access technologies such as Web Services, new computational models such as Grid computing, and new overlay networking methods such as peer-to-peer networks are creating the basis for purely distributed systems.

The ineluctable push toward *distribution*, *mobility*, and *ubiquity* poses new challenges to the management of such systems. All conventional approaches require a certain degree of centralized control, exerted via either a single management entity or an organized set of management entities. Hierarchical and distributed management approaches have become increasingly popular in network and systems management. However, when systems' size, dynamics and ubiquity are pushed to the limit, the obvious approach is to dissolve all management functions into the system itself, creating so-called *self-managing systems*.

Moving toward this direction, the concept of *autonomic systems* has been recently introduced. This refers to communication and/or computational systems that can "adaptively" *self-configure*, *self-optimize*, and *self-heal*. Autonomic systems do need to be able to operate autonomously and survive problems in the absence of any external management intervention. However, they still have to rely on some sort of "fabric" supporting fundamental services such as node and path discovery, service publishing, etc.

In practice, and for scalability reasons, autonomic systems adopt a *clustering* approach that involves a certain level of organization among the various entities (or nodes) of the system. A good example, in the area of networking, is offered by ad hoc networks, where *terminodes* are organized in clusters having a node with special functions, termed *cluster head* [1]. Similarly, peer-to-peer systems are structured in *peer groups*. Discovery, publishing, and internode communication are solved, again, via special nodes termed *rendezvous* or *super-peers* [2].

Efficient clustering is a fundamental problem in the area of networking and distributed services. Because of their peculiarities, autonomic systems are particularly sensitive to the way clusters are formed and cluster heads are elected. Assuming that autonomic ubiquitous systems are composed of entities that move, attach/appear and disconnect/disappear fairly frequently, autonomic systems must rely on effective means for maintaining clusters and cluster heads. This includes the ability to partition the system into an appropriate number of clusters (depending on the number of system entities) and elect the best possible cluster heads.

In this paper, we address the clustering problem for autonomic systems, presenting a novel approach that has the following advantages: 1) it is based on a distributed algorithm that has a low (linear) cost (efficiency); 2) it can satisfy precise constraints on the number of clusters (self-configuration); 3) it creates provably near-optimal clusters and cluster heads (self-optimization); and 4) it recalculates near-optimal cluster heads in face of component failures or congestion (self-healing).

To the best of our knowledge, no existing technique satisfactorily addresses the combined requirements of efficiency, scalability, adaptability, and optimality. Our contribution includes an in-depth simulation-based analysis of the proposed approach, elaborating on its applicability to distributed monitoring, peer-to-peer systems, network overlays, application-level multicast, and content adaptation networks.

II. Existing Approaches

Clustering algorithms have been the subject of intensive studies in many different disciplines. For brevity, we focus on their theoretical foundations originating in graph theory. The most relevant work has been carried out during the last

Manuscript received November 1, 2004; revised May 4, 2005. This work was supported in part by the U.K. Engineering and Physical Sciences Research Council (EPSRC) under Grant GR/S09371/02 (Polymics Project).

C. Ragusa is with the IT-Innovation Centre, University of Southampton, Southampton SO16 7NP, U.K. (e-mail: cr@it-innovation.soton.ac.uk).

A. Liotta is with the Department of Electronic Systems Engineering, University of Essex, Colchester CO4 3SQ, U.K. (e-mail: aliotta@essex.ac.uk).

G. Pavlou is with the Centre for Communication Systems Research, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: g.pavlou@surrey.ac.uk).

Digital Object Identifier 10.1109/JSAC.2005.857203

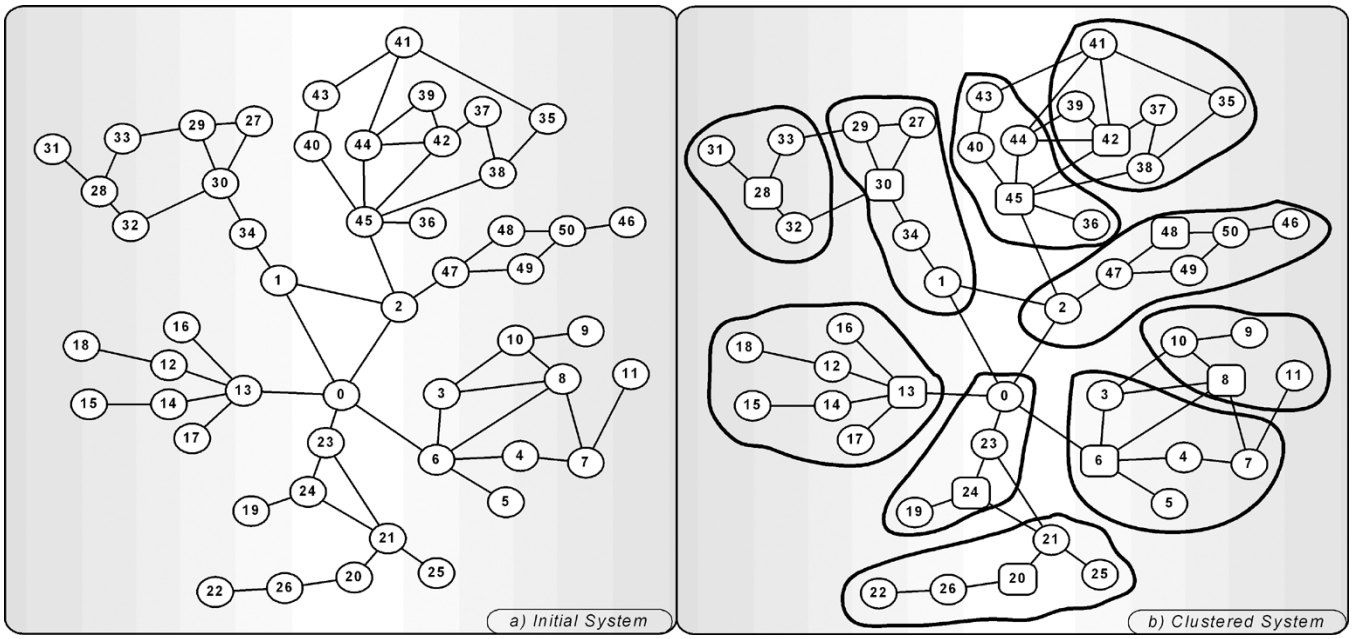

Fig. 1. (a) Example system topology. (b) Clustering and cluster heads (rounded boxes) resulting in minimal THD.

30 years under the banner of network location or partitioning. This involves the optimal placement of p service facilities in a network of N nodes, which is equivalent to creating p clusters and electing the optimal cluster head for each of them. Optimality is directly associated to the type of operation conducted by the cluster head, through the minimization of a given objective function. For example, the location problem is termed the *p-median* problem when the objective is to minimize the overall traffic incurred by the messaging among cluster heads and their respective cluster nodes. If the objective is to minimize the maximum response time within clusters (among cluster heads and cluster nodes), the problem is termed *p-center*.

The p-median and p-center problems are both NP-*hard* in general network topologies [3]. Numerous approximate polynomial algorithms have been proposed (see extensive survey presented in [4]–[6]) but none of them suits the aforementioned combined requirements of efficiency, scalability, adaptability, and optimality. We mention below some of the most prominent approaches to network partitioning/clustering.

In the class of enumeration algorithms, Hakimi presented a simple procedure for determining a single median (or one-median) in nonoriented networks [5]. The algorithm finds the median of a single cluster based on the distance matrix. Multiple optimal medians (clusters) are computed in polynomial time in [3]. However, their algorithm only works for tree networks.

If we look at graph-theoretic approaches, we also find unacceptable constraints. In this case, all existing algorithms take advantage of the underlying network structure to determine the cluster centres (e.g. [3], [7], and [8] work only for tree networks).

Algorithms valid for general networks are based on heuristic approaches, which rely on trial-and-error methods (a comprehensive review is provided by [4] and [6]). The limitation in this case is that it is not possible to establish the goodness (optimality) of the cluster head locations.

On the other hand, *Lagrangian relaxation* algorithms are optimization approaches based on relaxations of the integer-programming formulation of the *p-median* problem [4]. When coupled with one or more heuristic algorithms, Lagrangian approaches often give results that are provably optimal or near optimal. Nevertheless, Lagrangian algorithms are not viable for our clustering problem because they need to use the distance matrix.

The approach proposed in this paper belongs to the category of heuristic algorithms but is completely different from existing work. We exploit the key features of code mobility to achieve a fully distributed clustering algorithm that is also adaptive to network conditions. The code mobility paradigm has been broadly used in the past as an alternative to the client-server model because of its potential to tackle problems, where scale and dynamics assume significant importance [9]. Mobile software entities with particular properties such as autonomy, proactivity or ability to learn from the environment are commonly referred to as mobile agents (MAs). In the context of this paper, we adopt a combination of code mobility with some of the properties typical of MAs, such as cloning. However, our system does not require any heavyweight MA platform to operate, as discussed in Section X-A.

III. Problem Formulation

A logical/abstract view of our self-managed system is illustrated in Fig. 1(a), where nodes may represent anything from the software entities of a distributed system (or application) to networked devices. We consider nodes to be interconnected through Internet-based networking and transport protocols [i.e., transmission control protocol/Internet protocol (TCP/IP)].

Routing information is essential for our clustering approach and we assume that, typically link-state, routing algorithms operate independently from our system, while the latter has read-only access to the routing tables [e.g., via the simple network

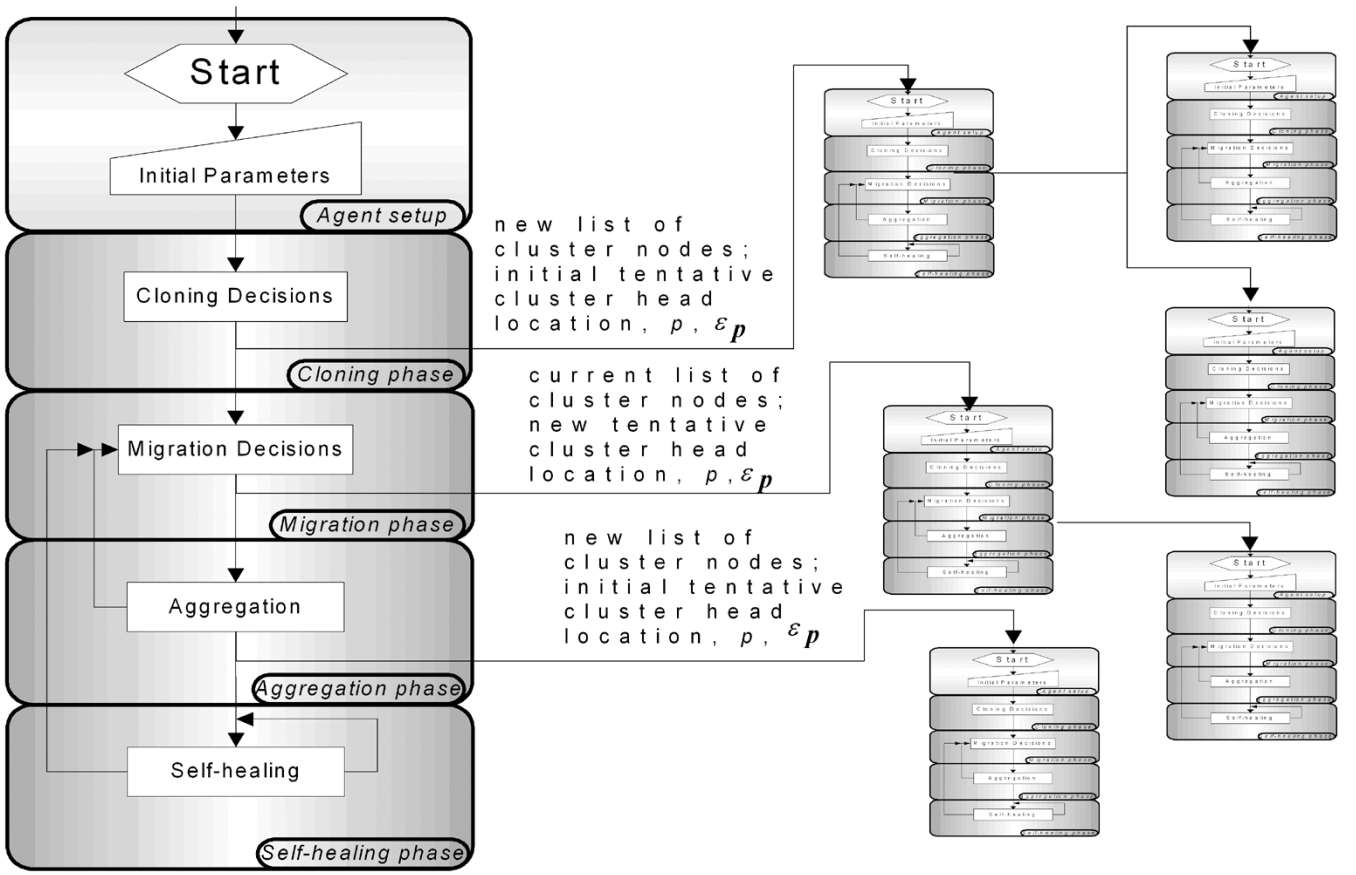

Fig. 2. Algorithm overview (the small diagrams on the right are identical, rescaled versions of the diagram on the left).

management protocol (SNMP) routing management information base (MIB)]. It is the responsibility of the routing algorithm to maintain up-to-date routing tables reflecting current network connectivity. The system is assumed to be fully dynamic, i.e., nodes are generally mobile, connectivity is intermittent, and new nodes can join or abandon the system at any time. The clustering problem can be formulated as follows.

Given a set of interconnected nodes (N) and an objective function (O), create p partitions (or clusters) and elect their cluster head nodes (one per partition) so as to minimize the objective function.

The main requirements on the clustering process are: 1) it must be scalable with network size; 2) it must be possible to define an upper bound on clustering time; and 3) cluster heads should be continuously reelected to maintain optimality as system conditions change (link failure, node mobility, etc.).

The remainder of this paper refers to a more specific clustering problem, the p-median problem. In this case the objective function is the *total hop distance* (THD) in the cluster, i.e., the sum of the distances between cluster heads and their respective cluster nodes, so that the cluster head is the most "central" node. Hence, the clustering problem can be formulated as follows.

Given a set of interconnected nodes (N), compute a near-optimal p-median that self-reconfigures itself according to system dynamics and self-heals in case of link or node failures.

Following again the example of Fig. 1(a), the aim is to create near-optimal clusters with dynamically reconfigurable cluster heads. If we assume, for simplicity, a constant message exchange pattern between cluster heads and cluster nodes, the configuration shown in

Fig. 1(b) results in minimal overall traffic. In face of topological changes, our system reelects cluster heads to maintain optimality (see Section IV-G)

IV. Self-Managed Clustering Through Mobile Agents (MAs)

A. System Overview

The general principle behind our approach is inspired by distributed routing algorithms, i.e., Dijkstra or Bellman–Ford, as used in IP networks. Nodes continuously compute routing tables based on very simple operations, resulting in a global, distributed database containing information that reflects the network status. In the remainder of this paper, we shall demonstrate how this information can be used by relatively simple heuristic procedures in order to solve the clustering problem formulated in Section III.

Our approach is to use a MA system that actually computes the clusters following the five-phase process depicted in Fig. 2. The process can start at any node (see Section VIII for our study on its independence from starting node) by injecting a single MA. The starting node is also the cluster head of the whole system to be partitioned, so effectively there is only one cluster initially that contains a list of N nodes. The agent is setup with a goal in the form of a simple heuristic procedure that drives the partitioning process. The input parameters are the following:

- list of nodes N to be partitioned (this is a list of node identifiers, e.g., IP addresses);
- number of target partitions p;
- tolerance margin on p, i.e., $\boldsymbol{\varepsilon}_p = ((\Delta p/p)) * 100$; ($\Delta p$ is the absolute variation of p);
- initial location of the cluster head i.

Starting from its initial location (i.e., the root node), the MA initiates an iterative, distributed procedure that subsequently creates new partitions and clones new agents (one per new partition). Every MA in the system holds exactly the same logic (i.e., they all go through the process depicted in Fig. 2, and they all behave in the same fashion). We provide below an overview of the five-phase procedure performed by each agent—further details are given in Section IV-B and onwards.

Our stratagem is to obtain the required clusters through a clone-and-migrate process. The agent performs a matching operation between the local routing table and the list of nodes included in its partition. Based on the status of the network and on the relative distance between cluster head and cluster nodes, the MA can take four different decisions.

1) **Cloning/partitioning**: If the cluster has a size considerably larger than $|N|/p$, the MA splits the partition in two or more clusters; clones one agent per partition; initializes them with their respective list of nodes; sets up their respective initial cluster head location; and spawns them.[1] Each new agent initiates the whole process from scratch, the only difference being the list of cluster nodes and the initial cluster head location.
2) **Migration**: If the cluster has a size comparable to $|N|/p$ it starts migrating within the cluster until the central point is found. The metric for centrality may vary—i.e., herein, we find the median.
3) **Aggregation**: In some cases, the cloning/partitioning process may not succeed in creating clusters of size comparable to $|N|/p$. This happens for instance when cluster nodes are sparsely distributed. In order to meet the requirements on the target cluster size, the agent creates an appropriate number of subclusters by subsequently aggregating nodes on the bases of their distance from the agent.
4) **Self-healing**: Once the clustering process is completed and the cluster head has been elected, the MA gets into self-healing mode. It periodically rechecks the network conditions to reenter the migration mode when these change (e.g., due to link failure, congestion, or node mobility).

[1]The operator $|.|$ returns the cardinality of a list of nodes, e.g., $|N|$ is the number of elements of N.

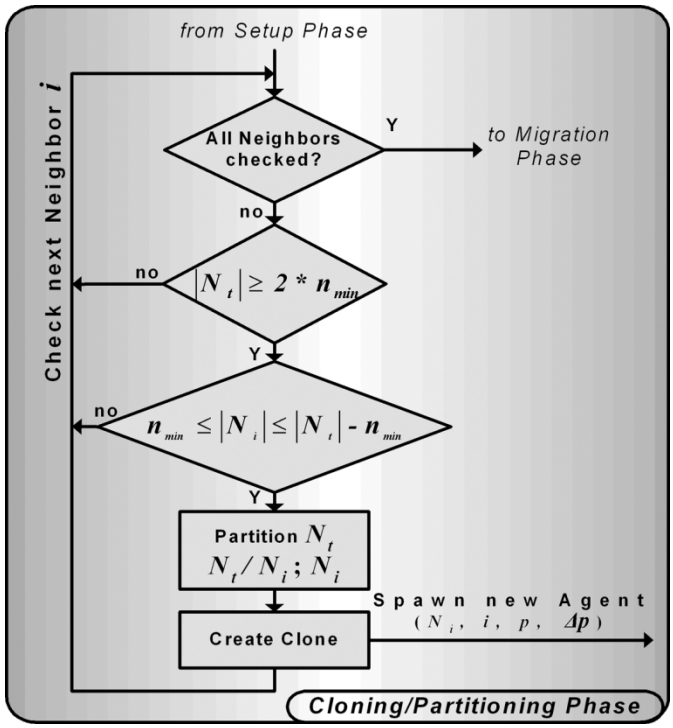

Fig. 3. Cloning phase flowchart.

The following sections, describe the agent behavior in greater details, including parameters, thresholds, and heuristics used in the MA decision process.

B. Parameters and Definitions

In this section, we specify the notation used in the remainder of the paper. Starting from the aforementioned input parameters, the agent computes the following variables during the setup phase.

- $\overline{n} = \text{round}(|N|/p)$ is the average number of nodes per partition.
- $\Delta p = (p * \boldsymbol{\varepsilon}_p/100)$ is the absolute variation admitted on p.
- $\overline{n}_{\min} = \text{round}(|N|/(p + \Delta_p/2))$ is the min value of $\bar{\mathbf{n}}$.

The following parameters are used in the cloning and migration phases.

- N_t is the list of cluster nodes managed by an MA. At bootstrapping time $N_t \equiv N$.
- N_i is the list of cluster nodes reachable through the neighbor node having identifier i.

C. Cluster Generation—Cloning Phase

The cloning process that results in the generation of new clusters is depicted in Fig. 3. At this point, the MA is sitting onto a certain node (the tentative cluster head), holds the list of cluster nodes, N_t, and has to decide how to partition them. By hashing N_t against the local routing table, the MA discovers the neighbor nodes—i.e., the next-hop addresses involved in the

communication between cluster head and cluster nodes. The decision on whether or not and on how to partition N_t is controlled by the three conditions specified in Fig. 3. The first condition makes sure that the main thread of the algorithm gets into the migration phase only upon considering all neighbor nodes. The second condition ($N_t \geq 2 * n_{\min}$) prevents cluster fragmentation. In fact, we do not want the partitioning process to result in partitions that do not have at least $n_{\min}$ elements. Also, we need to prevent the situation in which the original partition is depleted excessively by the partitioning process.

Having gone through the previous conditions, we know that the partition under scrutiny is too big. What we have to decide is whether or not the subpartition N_i is large enough ($|N_i| \geq n_{\min}$); but we also have to make sure that, should N_i form a new partition, the remaining nodes in N_t— i.e., $\{N_t/N_i\}$—would still constitute a sufficiently large cluster.[2] The latter holds when $|N_t| - |N_i| \geq n_{\min}$. If the third condition is verified, N_t is partitioned into two different partitions, $\{N_t/N_i\}$ and N_i. The main thread of execution continues the whole process using the former list and clones a new MA whose task is to take care of N_i.

Therefore, there are two types of exit conditions from the cloning phase.

1) One or more new MA clones/clusters are created, depending on the cloning conditions. Each MA is spawned to its initial location that is set to the relevant neighbor node i.
2) Once all branches of the routing tree rooted at the agent node have been assessed, the MA leaves the cloning phase and enters the migration phase, as explained in the next section.

D. Cluster Head Location—Migration Phase

Starting by its current location, the principle that drives an agent toward its central location (within its cluster) is based on the calculation of weighed routing costs (E_i) for each of the neighbor nodes (i), defined as $E_i = C_i * |N_i|/C_t * |N_t|$, where N_t is the list of cluster nodes of the agent, N_i is the subset of cluster nodes reachable through neighbor i, C_i is obtained by adding the individual routing costs of N_i (these are extracted from the local routing table), and C_t is the sum of all C_i. E_i provides an estimate of the global distance (in terms of routing costs) associated to each neighbor. The aim is to migrate toward the location with higher associated cost. So the agent elects as new candidate location, the neighbor i having maximum value of E_i (first box in Fig. 4).

Agents avoid migrating in loops by retaining the highest value of E_i for all previously visited nodes. So, as soon as the agent is presented with a candidate node that it had previously visited, it detects a potential looping condition. In that case, it simply elects the node having the highest value of E_i (among all historic values, including that of the new candidate node). At this point, we have two alternatives. If the newly elected node is in fact the node where the agent is sitting on, the agent goes to the aggregation phase (described below). Otherwise, it first disables

[2]The operator "/" returns the list of nodes resulting when removing list N_i from list N_t.

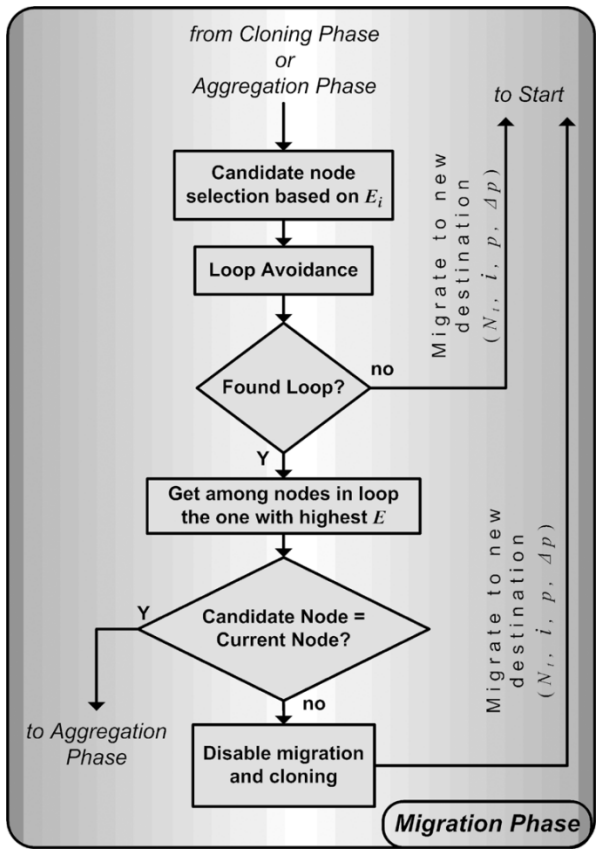

Fig. 4. Migration phase flowchart.

the migration and cloning flags before migrating. In this way, it makes sure that upon reaching the new location (and restarting the whole procedure), those phases are skipped, getting directly into the aggregation phase.

E. Cluster Consolidation—Aggregation Phase

We have seen how the cloning process is based on simple heuristics, which means that despite having a relatively simple and computationally efficient algorithm, we also have to cater for exceptional conditions. In practice, we have found rare occasions when networks having highly sparse nodes lead to a small number of clusters that are considerably larger than $|N|/p$ but still do not satisfy the cloning conditions of Fig. 3.

In order to meet the requirements on the target cluster size, the agent creates an appropriate number of subclusters by subsequently aggregating nodes on the bases of their distance from the agent (Fig. 5). Each newly formed partition sparks the cloning of a new agent whose cloning flag will be set to "disabled" for obvious reasons (the size of cluster already meets the requirements). On the other hand, upon exiting the partitioning loop, the main agent now has a minimal partition but should again get into migration mode to consider better locations.

F. Self-Clustering in Operation—An Example

To further illustrate the operation of the self-clustering algorithm, let us consider again the system topology depicted in Fig. 1(a). Since clustering is achieved via MA cloning/deployment, its steps can be highlighted by the MA deployment

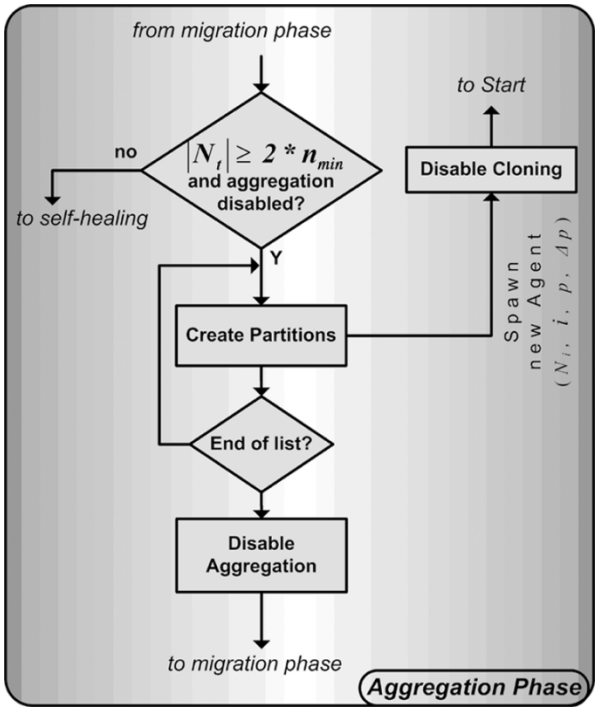

Fig. 5. Aggregation phase flowchart.

map depicted in Fig. 6, which leads to the clusters illustrated in Fig. 1(b). We are assuming that the process is initiated at node 0, where the first (unique) agent starts by assessing the cloning conditions of Fig. 3 for each of the neighboring nodes (nodes 1, 2, 6, 13, and 23). Given the input parameters $N = \{1, 2, \ldots, 50\}$, $p = 10$, and $\varepsilon_p = 20\%$, the cloning conditions are found to be true for all neighbors except node 23. Hence, four new agents are cloned, while the original agent migrates to node 23. At this point, each agent proceeds independently from each other. Agents have their own list of nodes—for instance, the agent sitting at node 1 has $N_t = \{1, 27, 28, 29, 30, 31, 32, 33, 34\}$. Let us now see what happens to this agent (other agents will behave analogously). The cloning conditions are not satisfied at this point because the agent is sitting too much in the periphery of the cluster. So the agent subsequently migrates to 34, 30, and 32. At 32, the cloning conditions are met, leading to the creation of two subpartitions, {1, 27, 29, 30, 34} and {28, 31, 32, 33}. The newly cloned agent subsequently migrates to 30 and 34. At this point, loop avoidance comes into play, leading to a final migration to 30. The other agent subsequently migrates to 28 and 31 and, following loop avoidance, settles at node 28.

G. Self-Healing Phase

Upon completing the whole clustering process, the agent system still has to continuously adapt to network dynamics. All agents gets into self-healing phase, consisting of a periodic recomputation of the weighed routing costs (E_i)—as described in Section IV-D. As network conditions change, so does $E_{\max}$—i.e., the maximum value of all E_i. As previously explained, $E_{\max}$ grows as a result of network degradation or failure; so by comparing the new value of $E_{\max}(E_{\max\_new})$ with its current counterpart $(E_{\max\_old})$, the agent can decide whether or not to trigger migration.

The periodic recomputation of $E_{\max}$ assumes the choice of an "appropriate" period τ. We recall that agents are meant to not only maintain location optimality but also perform the tasks of the cluster head. Consequently, there is no point in setting τ to be smaller than the typical agent migration time, risking otherwise the occurrence of MA system instability. *Per contra*, excessively high values of τ would compromise prompt adaptation. τ determines system stability, controllability and duration of transient conditions. However, the choice of τ is a tradeoff decision driven by the laws of classic control theory, so we do not delve into this aspect herein. More pragmatically, we have set τ to be twice the average migration time, obtaining good overall controllability.

V. Assessment Methodology

The proposed system has been evaluated through an extensive set of simulations that have provided an insight into *performance*, *scalability*, and *autonomy* (self-optimization, self-configuration, and self-healing). For brevity, only the most significant results are illustrated from Section VI onwards, focussing on computational complexity, sensitivity to starting point, optimality, and self-reconfiguration. Other nonreported results concern system stability. We have verified that the partitioning process always exits and that agent migration does not create oscillatory or looping conditions.

The simulation environment and the tools used for our study are depicted in Fig. 8, where shaded boxes indicate new software/tools developed by us. The proposed system is built on top of the NS network simulator [10], a widely used simulator within the research community, which supports the most common physical, link, network, transport and application layer protocols for fixed and mobile communication networks. We had to extend the NS core with the main MA functionality, including agent migration, cloning, destruction, and so forth. We have created a generic framework in which relatively complex MA systems could run over arbitrarily complex Internetworks. In this way, our partitioning method was studied under a large variety of conditions such as network topology type, network size, communication protocols, and so on. These conditions, or scenarios, are specified as Tcl scripts.

Another important aspect concerns the generation of network clusters, or topologies, that resemble real networks. Realistic Internet-like topologies where generated with the GT-ITM topology generator [11], following a well-established methodology by Calvert and Zegura [12], [13]. Transit-stub topologies having 51, 102, 147, 198, 258, and 303 nodes were generated in order to assess the algorithm sensitivity to network size. For statistical significance, we produced families of at least ten topologies each. When the variance of the results was not satisfactory we produced 20 topologies for family.

Since NS is an event-driven simulator, the analysis is carried out on the output traces. Given the large amount of simulations (the project took over three years worth of simulations), we had to automate the conversion of NS traces in a format that could be fed to statistical tools. We produced scripts in Tcl, Perl, and

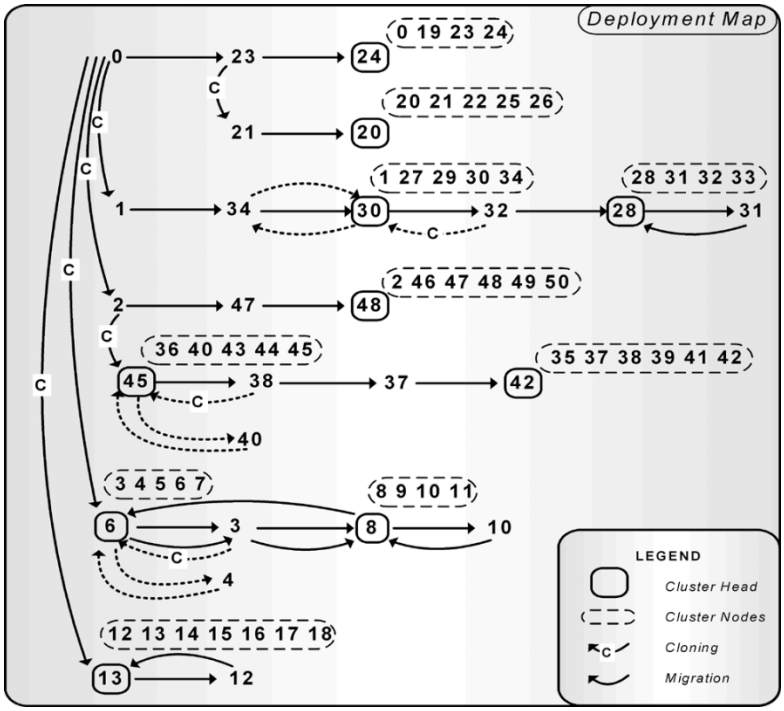

Fig. 6. Clustering process for the topology of Fig. 1(a), including the MA deployment map resulting in the optimal configuration of Fig. 1(b).

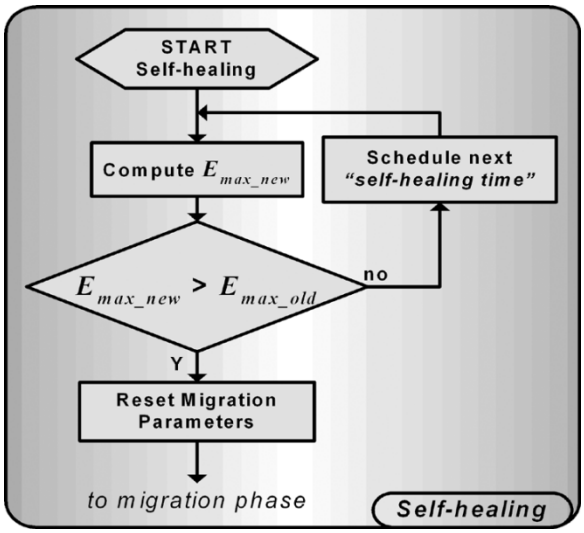

Fig. 7. Self-healing algorithm.

Matlab for this purpose. The data was analyzed using Microcal Origin [14]. The NAM network animator [15] was also used for debugging, verification, and validation.

Another important aspect was the study of the optimality of our approach. This was done by a comparison with the well-known *Lagrangian* algorithm [4], which computes provably near-optimal p-medians. Despite computing p-medians in polynomial time, the *Lagrangian* algorithm relies on the whole network topology (i.e., the distance matrix), so it is not a viable (scalable) solution in practice. It can only be used as a benchmark for other algorithms, like in our case. We used the SITATION simulation package [16] to compute *Lagrangian* p-medians and their total hop distance.

VI. Computational Time

An important requirement of the clustering problem defined in Section III is scalability, which is the ability to increase the size of the problem domain with a small or negligible increase in the solution's complexity. In our case, the main scalability factors are the size of the initial cluster $|N|$ the cluster diameter (i.e., the maximum distance between any two nodes in the cluster), and the number of target partitions (p). Initial simulations, not shown here for brevity, gave a clear indication that in practice p and $|N|$ were related and could be assimilated to a single factor—i.e., their ratio.

Our initial hypothesis was that, because of its distributed nature, the complexity of our algorithm would be affected mainly by network diameter. The cloning process of Fig. 3 guarantees a good level of parallelism, while the migration process of Fig. 4 stops, in most cases, well before visiting the whole network distribution tree. In order to assess our hypothesis and determine the computational time of our algorithm, we have computed the clustering time (i.e., the MA deployment time) in all combinations of the parameters depicted in Table I. Fig. 9 depicts the results obtained for a subset of the simulation cases. However, all other cases gave equivalent results. The most eminent result is linearity with network diameter, which makes the algorithm suitable for large-scale systems.

It may be worth mentioning that in all cases, the slope of the clustering line is relatively small in comparison to the intercept between the line and the y axis. The reason for that is that MA migration time dominates the clustering process. In most common general purpose MA platform, the time to hop between two nodes is in the order of hundreds of milliseconds. This means that our algorithm would not be able to compete with conventional partitioning algorithms in the case of small-scale networks. On the other hand, the linearity and relatively small

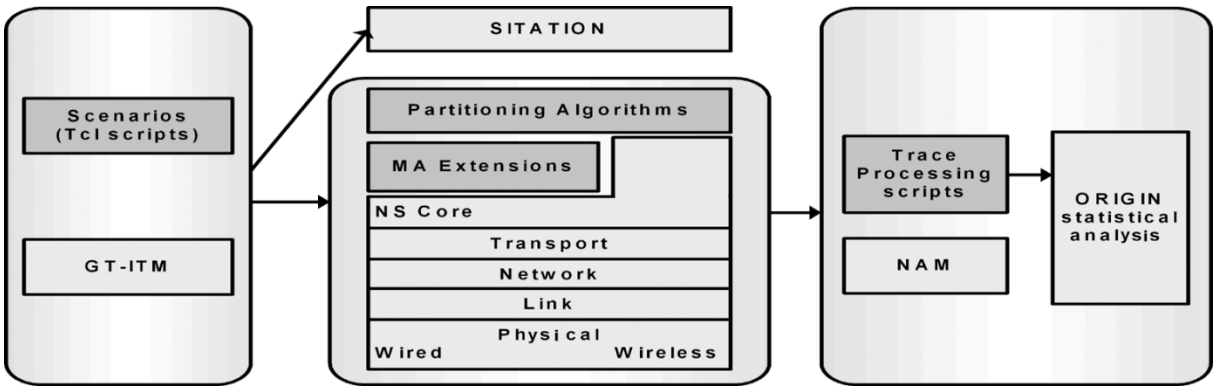

Fig. 8. Simulation environment and tools.

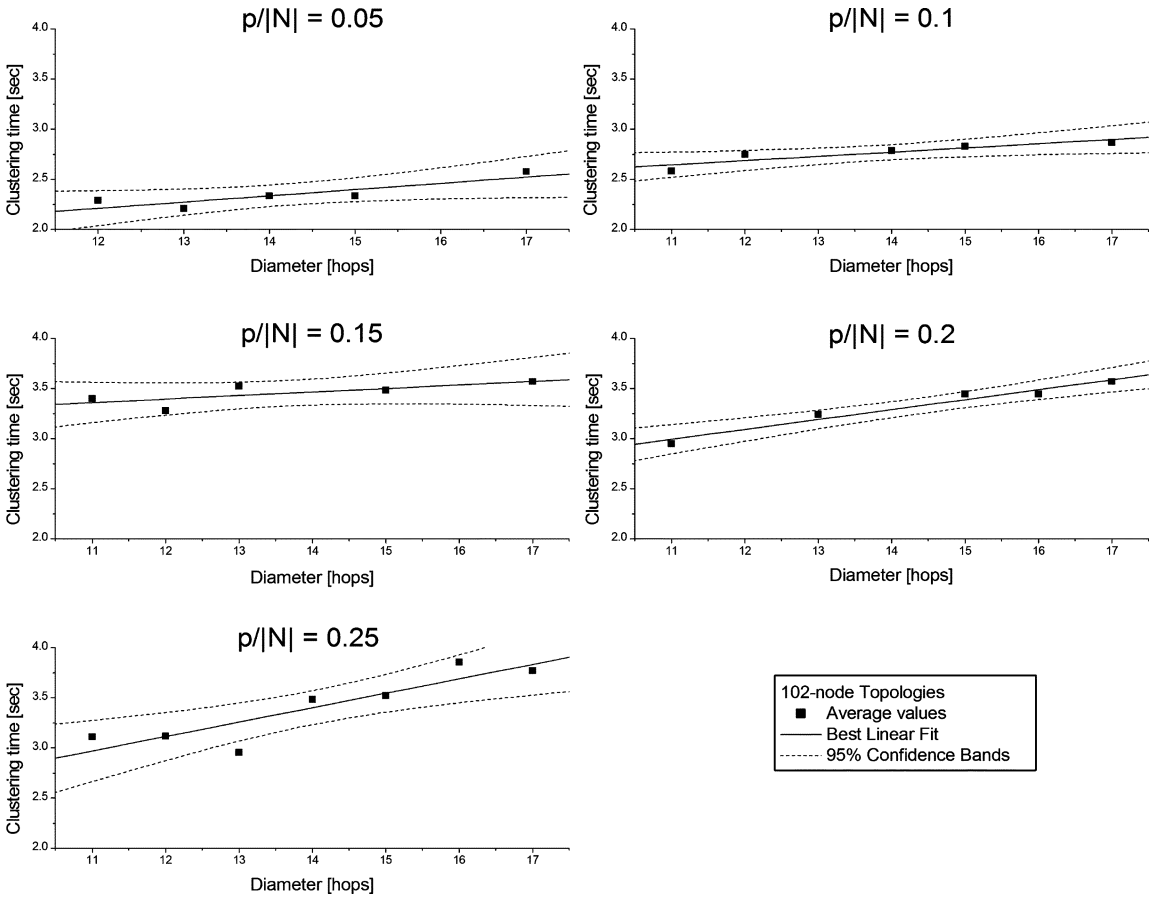

Fig. 9. Clustering time versus diameter for 102-node topologies.

TABLE I
SIMULATION DESIGN FOR THE STUDY OF SCALABILITY

Symbol	Quantity	Range
$\|N\|$	Topology size	102 147 198 258 303 (10-20 random topologies per family)
$p/\|N\|$	Clustering factor	0.05 0.1 0.15 0.2 0.25 0.3
ϵ_p	Percentage of p	15%
D	Network Diameter (hops)	11 12 13 14 15 16 17
R_{ID}	Root node ID	0 10 20 40 80

slope, ensure that the algorithm is well suited to very large-scale systems.

It may be worth mentioning that, in typical Internet-like networks, a small increase in network diameter results in a relatively large increase of $|N|$ especially for large networks. That is why when we studied clustering time versus $|N|$, we found that the former increases when an increase of $|N|$ is also accompanied by an increment in network diameter. In other words, an increase in $|N|$ with constant diameter, does not affect clustering time.

A second important set of simulation results is depicted in Fig. 10, which captures the effect that the number of target clusters has on computational time. In this case, we notice an even better behavior. Computational time gradually increases only in the lower range of $p/|N|$ but plateaus when the number of target

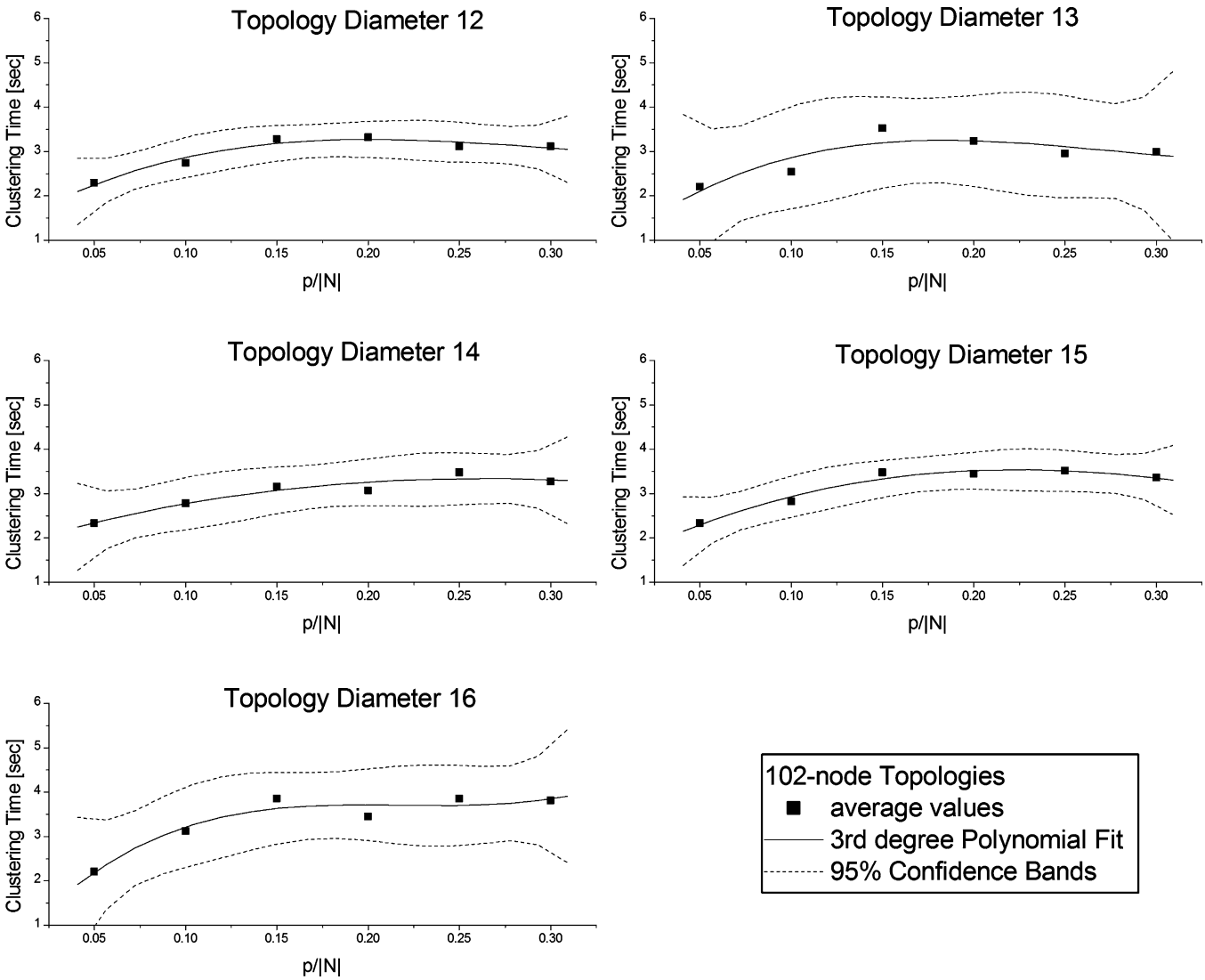

Fig. 10. Clustering time versus $p/|N|$ for 102-node topologies.

partitions is greater than about 15% of the initial cluster. This result shows, once more, the high level of parallelism and distribution of our algorithm. In fact, the more partitions we want to create, the larger will be the number of MA's operating in parallel.

VII. Accuracy

A. Location Optimality

Having verified the viability (scalability) of our clustering algorithm, we then studied its ability to produce "good" partitions and cluster heads, according to the problem formulated in Section III. We recall that our objective function is to minimize the THD—this is equivalent to saying that we need to assess the "distance from optimality" of the p clusters and cluster heads found by the algorithm. As mentioned in Section V, we have carried out a comparative study using the *Lagrangian* algorithm as benchmark. We have computed the THD of both algorithms in all combinations of the parameters depicted in Table II. Fig. 11 depicts just one of the cases, having $|N| = 51$ nodes. The most interesting and, to a great extent unexpected, result is that the THD of our algorithm has been found to be always smaller than the THD of the *Lagrangian* algorithm, which proves that our solution is al least near-optimal too. We should stress, again, that this means that we have a clustering algorithm that is computationally viable (the *Lagrangian* algorithm is not) and near-optimal.

TABLE II
Simulation Design for the Study of Optimality and Error on Computed Number of Partitions

Symbol	Quantity	Range
$\|N\|$	Topology size	51 102 147 198 258 303 (20 random topologies per family)
$p/\|N\|$	Clustering factor	0.05 0.1 0.15 0.2
ε_p	Percentage of p	0 5 10 15 20 25
D	Network Diameter (hops)	any
R_{ID}	Root node ID	0 10 20 40 80

B. Accuracy on Number of Clusters

Being based on simple heuristics, the algorithm will generally work better under certain conditions. Unexpectedly, the

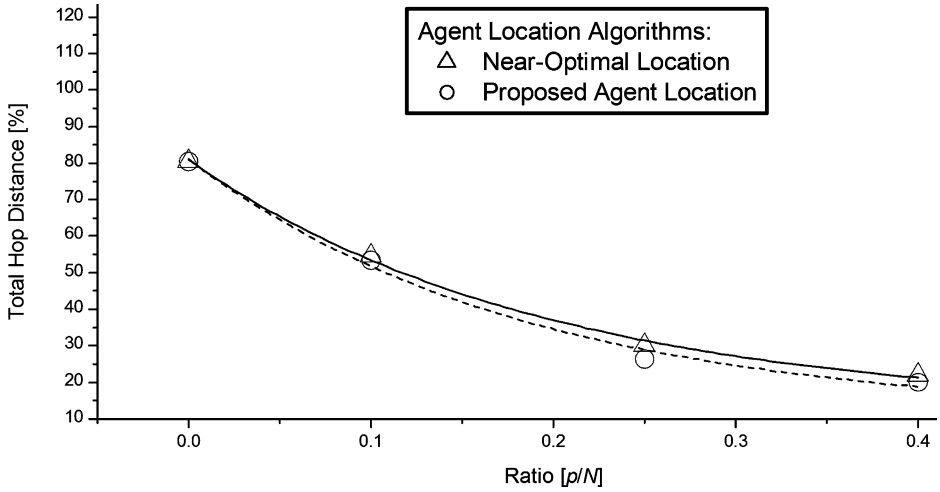

Fig. 11. Location optimality comparison.

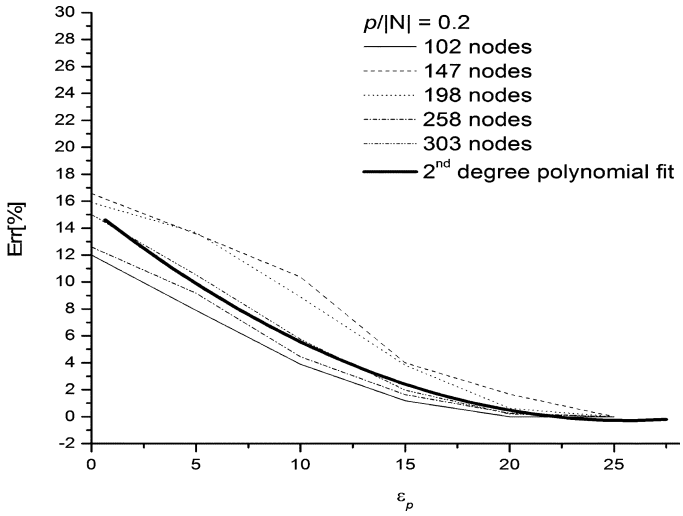

Fig. 12. Error on number of clusters produced by the algorithm.

algorithm has produced near-optimal p-medians under all tested cases (as discussed in the previous section). We also assessed its accuracy in the generation of the required number of clusters. We have seen that one of the input parameters is ε_p, which is the tolerance on p expressed in percentage. We have assessed the error on ε_p in all combinations of the parameters depicted in Table II, using the following formula:

$$\mathrm{Err} = \begin{cases} \frac{(p-\frac{\Delta p}{2})-p_O}{(p-\frac{\Delta p}{2})}, & \text{if } p_O < p - \frac{\Delta p}{2} \\ \frac{p_O-(p+\frac{\Delta p}{2})}{(p+\frac{\Delta p}{2})}, & \text{if } p_O > p + \frac{\Delta p}{2} \\ 0, & \text{if } p - \Delta p < p_O < p + \Delta p \end{cases}$$

where p_o is the actual number of clusters computed by the algorithm. A representative sample of all cases of Table II is depicted in Fig. 12. We can see that the accuracy increases considerably (i.e., Err diminishes) as we relax the constraints on tolerance on p (i.e., when ε_p increases). The algorithm is not particularly accurate in the generation of the exact number of clusters. It does compute near-optimal clusters, but it can only meet relatively relaxed constraints on the exact number of clusters. Nevertheless, in practical applications, the latter property is far less important than the former. One is most commonly interested in creating optimal clusters rather than determining their exact number *a priori*. What is typically needed in the applications described in Section X-B is the ability to partition a large system into smaller systems for efficiency reasons. Given the results of Fig. 12 (typical errors are of the order of 10%), we can conclude that the algorithm provides a good control of the order of magnitude of p.

TABLE III
SIMULATION DESIGN FOR THE STUDY OF SENSITIVITY TO STARTING NODE

Symbol	Quantity	Range
$\|N\|$	Topology size	51 198 (20 repetitions)
$p/\|N\|$	Clustering factor	0.05 0.1 0.2
ε_p	Percentage of p	15%
D	Network Diameter (hops)	any
R_{ID}	Root node ID	all

VIII. Sensitivity to Starting Conditions

Another important requirement of the algorithm is its independence from the input parameters specified in Section IV. The results reported below demonstrate that the algorithm provides comparable results in all the cases of Table III—we have computed clustering time, number of computed partitions, and THD. A representative sample of all results is illustrated in Fig. 13 that depicts mean values and standard errors. Noticeably, all mean values are comparable, demonstrating that the algorithm works equally well regardless of the starting node.

IX. Self-Healing

Large-scale, dynamic systems cannot rely on consistent assumptions on traffic, congestion or fault patterns. Being autonomic, our system has to be adaptive to loading conditions without the need of any manual intervention. We report here the results of our study of adaptation to network link failure, which is also indicative of its ability to adapt to network congestion, node mobility, and so forth. We have simulated all combinations of the parameters of Table IV. For each family of topologies, we

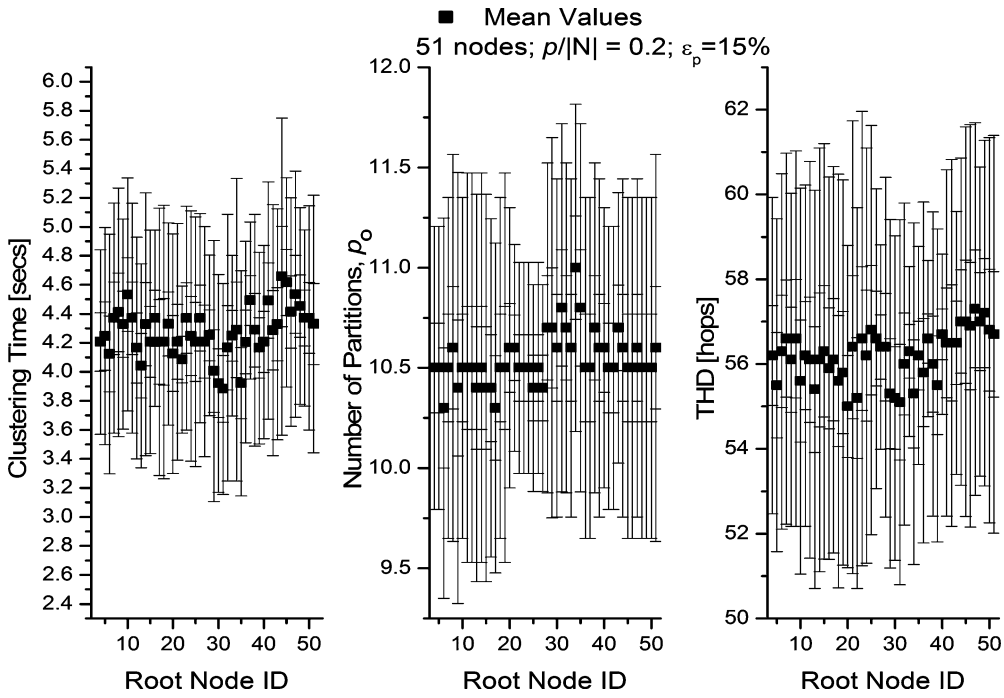

Fig. 13. Sample of results on sensitivity to starting node.

TABLE IV
SIMULATION DESIGN FOR THE STUDY OF SELF-HEALING

Symbol	Quantity	Range
$\|N\|$	Topology size	51 147 198 (20 repetitions)
$p/\|N\|$	Clustering factor	0.1 0.15 0.2
ε_p	Percentage of p	15%
D	Network Diameter (hops)	8 9 10 11 12
R_{ID}	Root node ID	0 10 20
$F_{\%}$	Percentage of faults	0 5 10 15 20 25

have randomly created an increasing number of link faults (from 0% to 25%)—for statistical significance, faulty conditions were randomized 20 times per topology.

Link failure does result into a change of topology—it is the responsibility of the IP protocol to update the routing tables accordingly. As the faults percentage increases, the probability to obtain unconnected nodes increases too. We have created a very large number of scenarios, selecting only the cases in which full connectivity is maintained. To benchmark the ability to self-heal, we have first measured the THD in the absence of any failure. Link failure results into new paths (between cluster heads and cluster nodes) that are generally equal or longer than the paths established before the failure. As a result, THD generally increases upon failure and the cluster head locations may become nonoptimal. To quantify the degradation in THD, we have measured its value immediately after link failure but just before triggering our self-healing procedure. We have also recalculated the theoretically near-optimal set of new cluster heads (using the Lagrangian algorithm) and, correspondingly, the new near-optimal THD. Finally, we have recalculated THD after self-healing, i.e., after MAs migrate to the newly calculated cluster heads. An indicative sample of the results is depicted in Fig. 14. Most eminently, self-healing succeeds to bringing the system back to optimality.

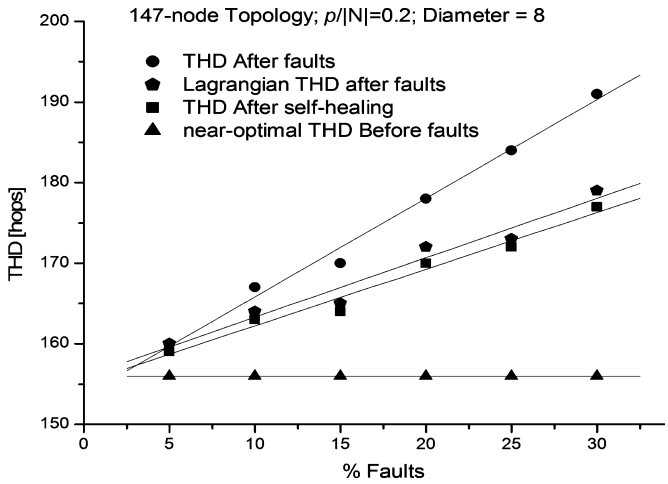

Fig. 14. Sample of results on self-healing.

X. DISCUSSION

A. Practical Deployment of the Algorithm

Our simulation-based study provides an insight into the viability of the clustering algorithm in the context of Internet-based networked systems—the simulation design described in Section V covers a broad range of cases. We discuss here other practical issues toward a full deployment of the algorithm, focussing on the infrastructure requirements.

The first important question regards the level of maturing of frameworks supporting code mobility. Our system assumes the presence of lightweight means for transporting and executing code throughout the network. However, it does not dictate code mobility support in every node—that assumption would be unrealistic. On the other hand, code transport and execution in routers is not problematic nowadays. The IETF Distributed Management (DISMAN) Working Group is studying

mechanisms for distributing scripts that can perform arbitrary tasks [17]. Also, several proposals of MIBs are underway. In particular, Script-MIB defines an SNMP-compliant MIB and a standard interface for code pushing and pulling [18]. The MIB supports arbitrary programming languages and makes no assumption about code formats. Code execution in network devices such as IP routers is also viable today. Just to mention an example, Cisco routers provide the ability to run Tool Command Language (TCL) scripts including SNMP MIB object access [19]. Public access to MIB objects is read-only but there are also security mechanisms allowing full manipulation.

In our simulations, we have assumed the functionality and typical performance figures of Script-MIB and TCL. The resulting size of our mobile code in TCL is in the order of 50 Kbytes, while the one-hop migration time is in the order of 400 ms. We have not specifically tackled the security and safety issues related to code mobility, given our relaxed requirements on security (our clustering algorithm merely needs read-only access to routing tables) and the ample literature available in the subject (the interested reader may refer to [20] and the references reported therein).

B. Applicability of the Algorithm

The adaptive clustering method proposed in this article has been formulated and addressed in a general way. Because of that, it is easy to think of a range of applications that may significantly benefit from its features. Our algorithm solves one of the most controversial graph theoretical problems—i.e., how to efficiently partition a dynamic, large-scale network, and find the central location of each subpartition. Thus, the algorithm may be directly applied to any of the classic network optimization problems such as the *optimal service facility location problem* [4].

Getting into more pragmatic applications, adaptive clustering is the basis of ad hoc networks, where the key features of autonomic systems (self-configuration, self-optimization, self-healing) are being extensively investigated. Ad hoc networks are self-managed and, as such, they have to deal with a small, as well as large, number of terminodes. Clustering is one way to give structure to the network, addressing scalability issues. Cluster head selection is another fundamental problem when dynamic conditions cause frequent changes in the network topology. Our MA system can be used as a self-organizing management "fabric" for ad hoc networks.

A similar self-management infrastructure is also needed in *peer-to-peer networks* or, more generally, in *overlay networks* [21]. The former may scale up to millions of nodes but are meant to be used by a virtually unlimited number of users. Some of the existing peer-to-peer frameworks such as JXTA [22] are open source and aim at becoming *de facto* standards. However, they do lack effective mechanisms for peer group management (the equivalent of our clustering problem) and super-peer election (the equivalent of our cluster head selection). Also, given the rapid shift toward user mobility, self-organization and self-healing are bound to become major issues in peer-to-peer systems. We are currently looking at peer-to-peer middleware for mobile systems, making use of the clustering algorithm described herein.

At the same time, overlay networks are gaining significant momentum because of their ability to complement existing networking protocols with new application-level solutions. Our MA deployment system may be considered as a way to create adaptive overlays, with the agent bearing also the logic of the application. For instance, the overlay network may be used to deploy a new *application-level multicast protocol* or to realize a *content distribution network*. We have recently started investigating the former, obtaining encouraging results [23]. Other relevant work is surveyed in [24] and [25].

Going more toward examples in the network management arena, we have recently demonstrated the use of an early version of our algorithm for the realization of an *adaptive distributed monitoring* system [26], illustrating its scalability and adaptability to network dynamics.

Finally, in the context of service management, the emerging area of *content adaptation networks* is a strong candidate for our approach. The problem in this case consists in the creation of application-level routing trees via network overlays. However, nodes have also media transcoding capability, allowing the delivery of a given media stream to a set of users having different requirements on the media format. Our future plans include the investigating of our clustering algorithm in the context of this revolutionary streaming approach.

XI. Conclusion

In this article we tackle one of the fundamental problems behind autonomic systems, the adaptive clustering problem. We propose a novel algorithm featuring computational efficiency (linearity with network diameter), self-configurability (independence from initial conditions), self-optimization (near-optimal clusters and cluster heads), and self-healing (adaptation to component failures or congestion). We have carried out an extensive simulation-based analysis that allows drawing definite conclusions on those features. The article includes also a discussion on a range of possible applications, from ad hoc networks to application-level networking and from network management to service management. We have also mentioned some of our ongoing work on application-level multicast, peer-to-peer middleware for mobile systems, and content adaptation networks. Clearly, the scope of autonomic systems is much wider so many other uses may be found.

The work presented here has given us ample opportunities to learn a number of lessons. Code mobility is certainly a potential paradigm that, by its very nature, suites a variety of requirements of current systems. These are increasingly mobile, volatile, dynamic, ubiquitous, and distributed. When combined with peer-to-peer systems and network overlays, MAs do offer powerful means for addressing classic NP-complete problems in a simple, though approximate, way (our work provides and example of such a system). On the other hand, the self-manageability obtained in our case through code mobility and peer-to-peer communication poses also new problems in terms of controllability, stability, and predictability of the system behavior. Systems such as ours are extremely difficult to assess. Prototype-based experimentation or field trials do not help because

the testbeds that would be needed are hardly available to research organizations. Mathematical modeling is extremely complicated and can only help in specific cases such as specific types of topologies. On the other hand, simulations provide a good insight but are time consuming and do not allow exhaustive studies. Some of our earlier results highlighted occasional instability problems which we then addressed satisfactorily.

REFERENCES

[1] C. K. Toh, *Ad Hoc Mobile Wireless Networks*. Englewood Cliffs, NJ: Prentice-Hall, 2002.

[2] D. Barkai, *Peer-to-Peer Computing*. Santa Clara, CA: Rich Bowles, 2002.

[3] O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems—Part 2: The P-medians," *SIAM J. Appl. Math.*, vol. 37, pp. 539–560, 1979.

[4] M. S. Daskin, *Network and discrete location*: Wiley, 1995.

[5] S. L. Hakimi, "Optimum distribution of switching centers in a communications network and some related graph theoretic problems," *Oper. Res.*, vol. 13, pp. 462–475, 1965.

[6] F. Buckley and F. Harary, *Distance in Graphs*. Reading, MA: Addison-Wesley, 1990.

[7] A. J. Goldman, "Optimal center location in simple networks," *Transport. Sci.*, vol. 5, pp. 212–221, 1971.

[8] D. W. Matula and R. Kolde, "Efficient multi-median location in acyclic networks," *ORSA/TIMS Bull.*, no. 2, 1976.

[9] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Trans. Softw. Eng.*, vol. 24, no. 5, pp. 342–361, 1998.

[10] K. Fall and K. Varadhan, *The NS Manual*. Berkeley, CA: LBNL, Univ. California. [Online]. Available: www.isi.edu/nsnam/ns/.

[11] E. Zegura. The GT-ITM Topology Generator. [Online]. Available: www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html

[12] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling Internet topology," *IEEE Commun. Mag.*, vol. 35, no. 6, pp. 160–163, Jun. 1997.

[13] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for internet topology," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 770–783, Dec. 1997.

[14] Microcal ORIGIN. [Online]http://www.originlab.com/

[15] J. Mehringer, *The Network Animator (NAM)*. Los Angeles, CA: Inf. Sci. Inst., Univ. Southern California, . [Online]. Available: www.isi.edu/nsnam/nam/.

[16] SITATION software. [Online]. Available: http://users.iems.nwu.edu/~msdaskin/BookSoftware.htm

[17] IETF Distributed Management (DISMAN) Working Group. [Online]. Available: http://www.ietf.org/html.charters/disman-charter.html

[18] J. Schönwälder and J. Quittek, "Secure Internet management by delegation," *Comput. Netw.*, vol. 35, no. 1, pp. 39–56, Jan. 2001.

[19] Cisco IOS Scripting with TCL. [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123newft/123t/123t_2/gt_tcl.htm

[20] J. Schönwälder, J. Quittek, and C. Kappler, "Building distributed management applications with the IETF script MIB," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 5, pp. 702–714, May 2000.

[21] L. L. Peterson and B. S. Davie, *Computer Networks*. San Mateo, CA: Morgan Kaufmann, 2003.

[22] C. Qu and W. Nejdl, "Interacting the Edutella/JXTA peer-to-peer network with web services," in *Proc. IEEE Int. Symp. Applicat. Internet*, 2004, pp. 67–73.

[23] C. Ragusa, A. Liotta, and G. Pavlou, "A scalable application-level multicast approach based on mobile agents," in *Proc. IEEE Int. Conf. Netw.*, Sydney, Australia, Sep. 28–Oct. 1 2003, pp. 197–202.

[24] A. El-Sayed *et al.*, "A survey of proposals for an alternative group communication service," *IEEE Netw. Mag.*, vol. 17, no. 1, pp. 46–51, Jan./Feb. 2003.

[25] M. Castro *et al.*, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *Proc. IEEE INFOCOM*, Mar.–Apr. 2003, pp. 1510–1520.

[26] A. Liotta, G. Pavlou, and G. Knight, "Exploiting agent mobility for large scale network monitoring," *IEEE Netw. (Special Issue on Applicability of Mobile Agents to Telecommunications)*, vol. 16, no. 3, pp. 7–15, May/Jun. 2002.

Carmelo Ragusa received the Laurea degree in computer and electronic engineering from the University of Catania, Catania, Italy, in 2001 and the Ph.D. degree in communication systems from the University of Surrey, Surrey, U.K., in 2005.

He is currently a Research Engineer at the IT Innovation Centre, University of Southampton, Southampton, U.K., where he works on grid computing. He has published two journal papers, one book chapter, and five conferences papers in the area of service management and distributed computing. His other areas of interest include software code mobility, distributed systems, and peer-to-peer networking.

Antonio Liotta (M'98) received the Laurea degree in computer and electronic engineering from the University of Pavia, Pavia, Italy, in 1994, the M.Sc. degree in information technology from the Polytechnics of Milan, Milan, Italy, in 1995, and the Ph.D. degree in computer science from University College London, London, U.K., in 2001.

He is currently a Senior Lecturer in Networking and Service Management at the Department of Electronic Systems Engineering, University of Essex, Colchester, U.K. He was with the University of Pavia, first as a Research Fellow in 1995, and then as a Guest Lecturer in 1996 when he was also a Visiting Researcher at Polytechnics of Milan and at Hewlett-Packard Laboratories, Bristol, U.K. He was with the University of Surrey, Surrey, U.K. first as a Research Assistant (2000), and then as a Lecturer in Networking and Service Management (2001–2004). He has published nine journal papers, two book chapters, and over 40 conferences papers in the area of service management, distributed computing, and advanced networking.

Dr. Liotta is a Chartered Computer Engineer and Member of the Professional Body of Electronic Engineers, Italy, since 1995; a Registered Practitioner of the U.K. Higher Education Academy since 2004; and a Member of the Board of Editors of the *Journal of Network and Systems Management* since 2002. He has served the Technical Programme Committee of IEEE NOMS, DSOM, and IM since 2001. He is also a TPC member of several other international conferences.

George Pavlou (M'95) received the Diploma degree in electrical and mechanical engineering from the National Technical University of Athens, Athens, Greece, and the M.Sc. and Ph.D. degrees in computer science from University College London, London, U.K.

He is currently a Professor of Communication and Information Systems at the Centre of Communication Systems Research, Department of Electronic Engineering, University of Surrey, Surrey, U.K., where he leads the activities of the Networks Research Group. He has previously been a Senior Research Fellow and Lecturer in the Department of Computer Science, University College London, where he led research activities on network and service management. He has published 34 journal papers, around 100 international refereed conference papers, and has contributed to four books. His research interests focus on network management, networking, and service engineering.

Prof. Pavlou is a Chartered Engineer and Member of the Technical Chamber of Greece since 1984. He is on the Editorial Board of the IEEE eTRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, the *IEEE Communication Surveys and Tutorials* and the *Journal of Network and Systems Management*. He is Network and Service Management Series Editor of *IEEE Communications Surveys*. He is also a Technical Program Committee member of all the major conferences in network and service management and has been the Technical Program Chair of IEEE Integrated Management (2001).