Self-Aggregation Algorithms for Autonomic Systems

Elisabetta Di Nitto, Daniele J. Dubois, Raffaela Mirandola
Dipartimento di Elettronica e Informazione
Politecnico di Milano
via Ponzio 34/5, 20133 Milano, ITALY
(dinitto,dubois,mirandola)@elet.polimi.it

ABSTRACT

One of the today issues in software engineering is to find new effective ways to deal intelligently with the increasing complexity of distributed computing systems. In particular, one of the aspects under study in the field of autonomic computing concerns the way such systems can autonomously reach a configuration that allows the entire system to work in a more efficient and effective way. In this paper we investigate how it is possible to obtain *self-aggregation* of distributed components. We have used existing self-aggregation algorithms as a starting point, and, after an analysis phase, we have discovered some aspects that could be improved. Finally we have derived new algorithms that showed improved self-aggregating performances in most of the situations.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; J.m [**Computer Applications**]: Miscellaneous

Keywords

Autonomic computing, distributed and self-adaptable systems, clustering algorithms, performance analysis.

1. INTRODUCTION

One of the today issues in software engineering is to find new effective ways to deal intelligently with the increasing complexity of distributed computing systems. This is particularly important in a pervasive context where the environment is instrumented with devices of any kind that are able to communicate over a network in order to solve several types of problems and to offer various kinds of services to their final users.

Autonomic Computing [15] applied to pervasive architectures is trying to show that adding autonomic reasoning to each computational element in the system could simplify its management and reduce costs [24].

Bionetics '07, December 10-13, 2007, Budapest, Hungary.

In this field, researches are borrowing some ideas from the biological world [23]. In particular, they study the behavior of colonies of insects and their capability to *self-organize* [9]. The main goal is to apply similar capabilities to software systems of interconnected components that singularly, like ants for their anthill [3], have limited information and reasoning power, but, all together, contribute to the high-level goals for the whole system. Using this approach many complex problems can be solved by executing simple rules locally to each component of the system, regardless system size and without the need of a centralized control [6].

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

This paper aims at analyzing and understanding the "magic" that is beyond existing approaches to pervasive self-aggregation techniques, and at creating new techniques that are more efficient and effective in specific cases.

The organization of the paper is as follows. Section 2 describes the aggregation problem and presents some distributed algorithms that address it. Section 3 describes our improvements to the existing algorithms. Section 4 presents a performance analysis and highlights the advantages of the algorithms we have defined. Section 5 presents an overview of the state of the art. Finally, Section 6 concludes the paper.

2. SELF-AGGREGATION ALGORITHMS

A typical environment in which self-aggregation can happen is a network of interconnected entities called nodes. Each node is characterized by a *type* and by a list of nodes called *neighbors*. In this situation self-aggregation is the capability of each node to modify the connections with its neighbors in order to reach a more efficient and effective configuration.

In a real network, a node can be any piece of software that is able to communicate with the others, its type can be defined in various ways all aiming at allowing a node to recognize its similarity with respect to a specific application. Thus, the type can correspond, in an object-oriented style, to the set of services the node can provide (i.e., to its interface), or, in an agent-oriented style, to the goal a node is able to achieve, or to any combination of them. Connections (*links*) between neighbor nodes correspond to the ability of

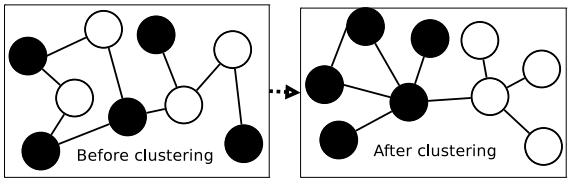

Figure 1: The effect of clustering on a set of nodes. Colors indicate the type of nodes.

a node to know the others, and is some case, it can result into direct physical connections established between pairs of neighbors.

The final purpose of a self-aggregation algorithm is to reduce the number of links from *incompatible* nodes and to add new links to compatible nodes, where the notion of compatibility is related to the types of the nodes. Here we study two simple cases of self-aggregation:

- *Clustering* (or *normal clustering*): in this case nodes tend to establish links with neighbors of the same type. Therefore, compatibility between nodes is defined as the equality of their types. In a real network this can be useful if the nodes want to balance their workload [17].

- *Reverse-clustering*: in this case nodes tend to establish links with neighbors of a different type. Thus, compatibility is defined as the inequality of nodes types. This can be useful to build a community of nodes able to cooperate to execute and delegate difficult tasks.

Figure 1 shows the situation of a network of two types nodes (depicted by the black and white colors) before and after the execution of the clustering algorithm: as it can be seen from the picture, the network tends to organize in two areas, one containing nodes of type black and the other nodes of type white. Clustering algorithms, per se, are not new in the literature (see the state of the art analysis presented in Section 5). Here the interesting aspect is that clustering is not executed by a centralized entity, external to the network. Instead, it is executed in a distributed way thanks to the ability of each node to take a simple "disconnect/maintain the link" decision on the basis of the type of each neighbor.

Our analysis of such distributed algorithms has started from those developed by Fabrice Saffre for the CASCADAS project [16], with the objective of improving, if possible, their effectiveness and performances.

The algorithms support both clustering and reverse clustering, but for the sake of space we focus here mainly on the clustering ones. Two different *algorithm modes* have been envisaged to achieve the same result: *passive clustering* and *active* (or *on demand*) *clustering*. In the following paragraphs we will describe all of them.

2.1 Passive Clustering

The definition of Passive Clustering is presented in Algorithm 1. The following is an explanation of each iteration:

- a random node of the network elects itself as the matchmaker node;

Algorithm 1 Saffre Clustering Algorithms

Passive Mode

```
matchmaker = LOCALNODE
for i=1 to NUM_ITERATIONS
do
  if (matchmaker has neighbors n1 and n2
       such that n1 is compatible with n2) then
     add link between n1 and n2
     remove link between matchmaker and n1
  fi
od
```

Active Mode

```
initiator = LOCALNODE
for i=1 to NUM_ITERATIONS
do
  if ((initiator has neighbor matchmaker) and
      (matchmaker has neighbor n1 such that n1
       is compatible with initiator)) then
    add a link between initiator and n1
    remove a link between matchmaker and n1
  fi
od
```

- the matchmaker node chooses two neighbors that are compatible to each other and makes them establish a new link;

- the matchmaker removes a link between itself and one of the chosen neighbors.

The author's analysis in [16] found that this algorithm has a side effect: nodes with more neighbors have a greater probability to be chosen and earn a new link after an algorithm iteration. This side effect leads to the creation of supernodes in the network that may become points of failures. To solve this problem a variant of this algorithm has been proposed by its author.

2.2 Active (or on demand) Clustering

The Active Clustering is a variant of the first algorithm that solves the side effects of the previous algorithm, but requires more communication between nodes. The following is an explanation of each iteration:

- a random node elects itself as the initiator node and elects a matchmaker node among its neighbors;

- the matchmaker node chooses one neighbor that is compatible with the initiator and makes them establish a new link;

- the matchmaker removes a link between itself and the chosen neighbor.

3. PROPOSED ALGORITHMS

The algorithms presented in the previous section show that by using simple local rules it is possible to reach the high level goal of grouping together nodes of the same type (or of different types if we used the reverse clustering approach). The performance of these algorithms has been evaluated in [16] and clearly shows that the system tends to reach a steady state. Starting from these results we have tried to understand why such simple and local laws are able to organize complex networks in order to investigate possible

Algorithm 2 Fast Clustering Algorithms

Passive Mode

```
matchmaker = LOCALNODE
for i=1 to NUM_ITERATIONS
do
  if ((matchmaker has neighbors n1 and n2
       such that n1 is compatible with n2) and
      (matchmaker is not compatible with n1)) then
     add link between n1 and n2
     remove link between matchmaker and n1
  fi
od
```

Active Mode

```
initiator = LOCALNODE
for i=1 to NUM_ITERATIONS
do
  if ((initiator has neighbor matchmaker) and
      (matchmaker has neighbor n1 such that n1
       is compatible with initiator) and
      (matchmaker is not compatible with n1)) then
    add a link between initiator and n1
    remove a link between matchmaker and n1
  fi
od
```

Algorithm 3 Accurate Clustering Algorithms

Passive Mode

```
matchmaker = LOCALNODE
for i=1 to NUM_ITERATIONS
do
  if ((matchmaker has neighbors n1 and n2) and
     (matchmaker is not compatible with n1)) then
      add link between n1 and n2
      remove link between matchmaker and n1
  fi
od
```

Active Mode

```
initiator = LOCALNODE
for i=1 to NUM_ITERATIONS
do
  if ((initiator has neighbor matchmaker) and
      (matchmaker has neighbor n1 such that n1
       is not compatible with matchmaker) and
       (n1 != initiator)) then
    add a link between initiator and n1
    remove a link between matchmaker and n1
  fi
od
```

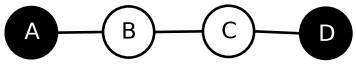

Figure 2: Particular network that can be clustered only by the accurate algorithm.

further optimizations. The most relevant thing we can note is that the previous algorithms not always perform operations that increase the number of links between compatible nodes. In this case it is said that the algorithm introduces some noise into the system. We will show that by trying to either limit or increase the level of noise within a network we can improve some aspects of the overall performances of the algorithm.

3.1 The Concept of Noise

The *algorithm noise* or randomness occurs when in the same algorithm iteration a new link is added between compatible nodes and then a link between compatible nodes is removed. In [10, 19] it is explained that in the biological world this noise is necessary to obtain an optimal solution, therefore it is reasonable to investigate the effects of an increase or a decrease of noise in the original self-aggregation algorithms.

3.2 Noise Reduction: Fast Algorithm

The first investigation we have done was to remove all the noise from the original algorithms: this resulted in a new algorithm (see Algorithm 2) that we call *Fast Algorithm* that is similar to the original one, but with the additional constraint that an algorithm iteration can never remove a link between compatible nodes. From the preliminary simulations we have seen that with respect to the original algorithm this one has a faster convergence rate because it avoids "noisy" iterations, another advantage is that it reduces the total number of link exchanges because of the lower number of neighbors that can be chosen. The disadvantage of this approach is that the increase in the number of links between compatible nodes is not as good as the original clustering. This means that the noise is a key factor for the accuracy of the algorithm.

3.3 Noise Increase: Accurate Algorithm

The second investigation we have done was to increase the algorithm noise in the following way (see Algorithm 3): the decision of adding and removing links for each algorithm step is fully unconstrained, except for the fact that the total number of links must remain the same and that a link between incompatible nodes can be added only if a link between incompatible nodes is removed in the same iteration. This constraint ensures that the aggregation of the system in the worst case remains constant and will never decrease. After the preliminary simulations we have seen a lower convergence rate and a larger number of exchanged messages with respect to the original algorithm, however the number of links between compatible nodes has been increased. This strategy is similar to what happens in genetic algorithms [14]: in a genetic algorithm each iteration has a mutation operation that randomly modifies the solutions that are computed until that moment. This prevents the genetic algorithm to get stuck in local optima and therefore improves the accuracy. Intuitively if we consider the situation in Figure 2, only a "noisy" operation (for example adding a link between incompatible nodes A and C) makes possible to add later a link between A and D and achieve the optimal level of clustering.

3.4 Adaptive Algorithm

This algorithm is a self-adaptive version of the previous algorithms with the aim to modify its behavior according to some local rules. These local rules have been modeled as a Finite State Machine (FSM). The general logic is that the algorithm starts behaving as the most constrained algorithm (Fast Algorithm) and stays in that state until the constraints inhibit further iterations (this happens when a node gets stuck because it does not have neighbors to choose that satisfy the algorithm requirements). In such a case the algorithm switches to a medium constrained algorithm (Original Saffre Clustering) first and then to the less constrained algo-

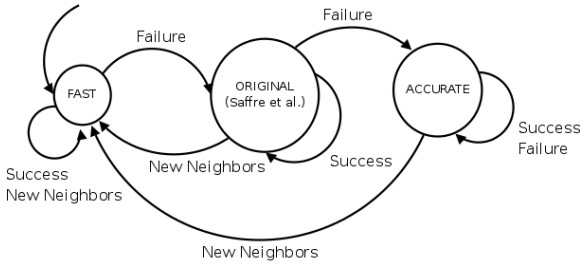

Figure 3: Adaptive Clustering Algorithm FSM

rithm (Accurate Clustering) if it gets stuck again. Finally, as soon as a new neighbor is added in a local node, it switches again to the most constrained algorithm.

The FSM in Figure 3 represents the current algorithms as states and the events as transitions. *Failure* transition is triggered when an algorithm is not able to complete an iteration because of its constraints:

- the matchmaker node does not have neighbors that are compatible to each other in original clustering;
- the matchmaker node does not have neighbors that are compatible to each other and whose one of them is not compatible with the matchmaker.

Success transition is triggered when an iteration terminates successfully.

New Neighbors transition is triggered when a new neighbor has been added to the local node.

4. PERFORMANCE ANALYSIS

In this section we will study the behavior of the proposed algorithms in different situations in order to identify their fields of applicability. We first describe the experiments set up and then present the results we have obtained.

4.1 Setting up the Experiments

This paragraph will discuss all the steps that are needed to study the performances of the self-aggregation algorithms that have been proposed in the previous section.

4.1.1 Methodology

The performances of the algorithms have been evaluated using the simulation framework for self-organization algorithms that has been developed in [13]. All the data that will be presented in this section have been obtained using Monte Carlo simulations. Each simulation has been repeated at least 20 times to provide some statistical validity, in addition, some preliminary simulations have been performed in order to identify which input parameters and performance indexes to consider when setting up the definitive thorough simulations. From the preliminary simulations we have also seen that the various algorithms do not show different behaviors if they run over 100 seconds, therefore we have chosen this number as the standard fixed duration for all the experiments. The most relevant parameters are described in the following paragraphs.

4.1.2 Input Parameters

The input parameters are determined by the environment in which the algorithms are run and by the algorithm itself. The bound values for these parameters have been chosen by observing the minimum/maximum values that have produced significant changes in the algorithms results of the preliminary experiments. The following is a list of environment-dependent parameters.

Number of nodes of the network: this is the fixed number of nodes of the network. During the simulations we have considered networks of 100, 200 and 300 nodes in order to obtain results comparable to the results that can be found in [11].

Number of types for all the nodes: all the network nodes will be differentiated using a uniform distribution of types. The least mixed network we considered has only 2 types, that is the minimum. The most mixed one has 15 types, that, over a population of 300 individuals, represents an average variety of 20 individuals per type. We also considered some intermediate number of types (5 and 10) to show what happens between the two bounds.

Average number of links for each node: the average number of links is stated at the beginning of the simulation and remains constant during all the experiment. The values that have been chosen for this parameter are 3 links, 4 links, and 5 links. Values under 3 links are not interesting because they produce topologies that tend most of the time to have too many nodes without links, while with values greater than 5 there is a tendency of creating a few super-nodes that group in their neighbors all the others.

Initial topology: this is the strategy that states how the initial links are established. With this parameter we wanted to create an initial pattern of interconnections that is similar to what we can find in different types of real networks. The topologies we have considered are:

- *Random topology*: all the links are created in a totally random way;
- *Torus topology*: all the nodes are connected to each other using a donut-like pattern;
- *Scale-free topology*: all the nodes are connected to each other in such a way that the probability of a generic node to be connected to k nodes is $P(k) = k^{-\gamma}$ where γ is a generic constant between 2 and 3 for most real networks [5].

The following additional input parameters are the algorithm-dependent parameters:

Algorithm mode: can be *active* or *passive.*

Maximum neighbor limit constraint: prevents the algorithm from adding new neighbors to nodes that have reached the maximum neighbor limit. This constraint is useful only when using the passive mode because this is the only mode that makes "rich" nodes become "richer" [21].

Definition of compatibility: in *normal clustering* two nodes are defined compatible when their type is the same, while in *reverse-clustering* two nodes are defined compatible when their type is different.

4.1.3 Performance Indexes

The performance indexes are the output parameters of the simulations we have done. Their purpose is to investigate about which goals the algorithm can reach in different situations and how well they can be reached. The following is a list of performance indexes that have been calculated and presented after the simulation process.

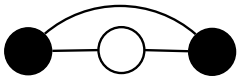

Figure 4: Example where a 100% homogeneity is not reachable

Homogeneity. This performance index, defined in [16], represents the aggregation of the network. This index is a number between 0 and 1: lower values mean a reverse-clustered network, while higher values mean a clustered network. The definition of this index is the following:

$$H = \frac{\sum_{i=1}^{N} v(node_i)}{L}$$

Where N is the total number of nodes, *v(x)* is the total number of nodes of the same type linked to x and L is the total number of links in the system. A problem of this index is that its bound values of 0 and 1 are not always reachable by simply moving links among nodes because of the structural characteristics of the network. Therefore it does not always give information about how far a generic self-aggregation algorithm is from its goal. This can be easily seen in Figure 4: the homogeneity is 67% and there is no algorithm that is able to improve this value using simple link movements.

Optimality. To solve homogeneity-related problems a new performance index called optimality has been defined. This index aims to be algorithm-related and it is a number between 0 and 1 like homogeneity, with the difference that its upper bound is always theoretically reachable. Reaching a value of 1 for optimality means that a clustering algorithm reaches the upper bound for the homogeneity, given the structure of the network. The formula that calculates optimality for a clustering and reverse-clustering algorithm is the following:

$$opt_{clustering} = \frac{H}{maxH},\ opt_{reverseclustering} = \frac{1-H}{1-minH}$$

Where H is the homogeneity, $maxH$ is the upper bound for homogeneity calculated by the simulator using a centralized optimal clustering algorithm, $minH$ is the lower bound for homogeneity using an optimal reverse-clustering algorithm.

Links-variance. This index gives information about how the node degree differs among the nodes.

$$linksv = \frac{\sum_{i=1}^{N}(d_i - \frac{2N}{L})^2}{N}$$

Where N is the total number of nodes, L is the total number of links, and d_i is the number of neighbors of node i. A high value of this index means that there are many super-nodes, a value of zero means that all the nodes have the same number of neighbors. A low value for this index is usually preferable, unless the logical super nodes correspond to the physical high capacity nodes.

Number of Messages. This index is directly computed by the simulator and gives an idea about how many messages are needed by an algorithm in order to achieve a goal. An example of goal can be the achievement of an 80% optimality in the case of normal clustering.

4.2 Results

What we are going to present now are the results of the experiments that have been run. For each experiment we will compare the various clustering approaches with a theoretical interpretation of each result. All results and charts of the experiments (also the ones that have not been included here) can be found in [1].

4.2.1 Reference Experiment for Clustering

In order to see how the algorithm performances are affected by changes of the input parameters, we have performed a particular experiment that has been used as reference. In this experiment we have run all the algorithms using values for the input parameters with the property of being the mean value between the bound values that have been chosen in section 4.1.2. The simulation results can be seen in Figures 5-8. In each figure the various algorithm variants have been grouped into two different charts in order to make the various curves more readable.

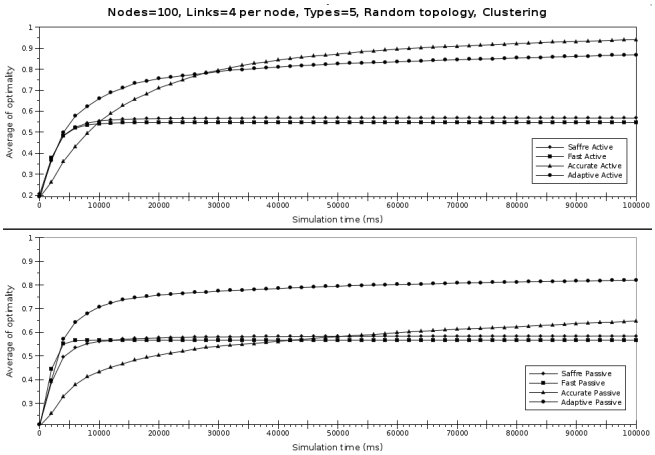

Figure 5: Reference Experiment: optimality average.

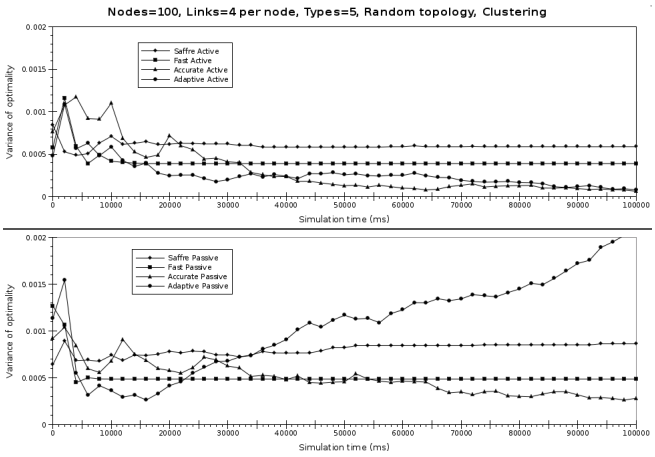

Figure 6: Reference Experiment: optimality variance.

The results in Figure 5 show how the noise can affect the convergence of the algorithms. In the fast algorithm a local

maximum is reached quickly with less messages, but then the homogeneity does not improve beyond that value. In the accurate algorithm local convergence is slow and requires more messages because of the noisy iterations, but then the algorithm convergence goes close to the global maximum. In this situation the adaptive algorithm tries to get advantage of both local and global convergence of the previous algorithms, therefore it seems a good choice in the average case if we want to improve homogeneity. On the other hand if the number of messages is more important than homogeneity, using the passive version of the fast algorithm is preferable because the passive protocol uses less messages than the active one (see algorithm 2).

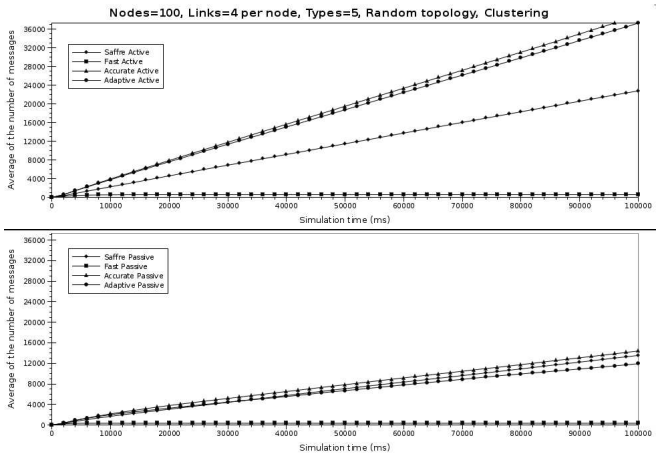

Figure 7: Reference Experiment: number of messages.

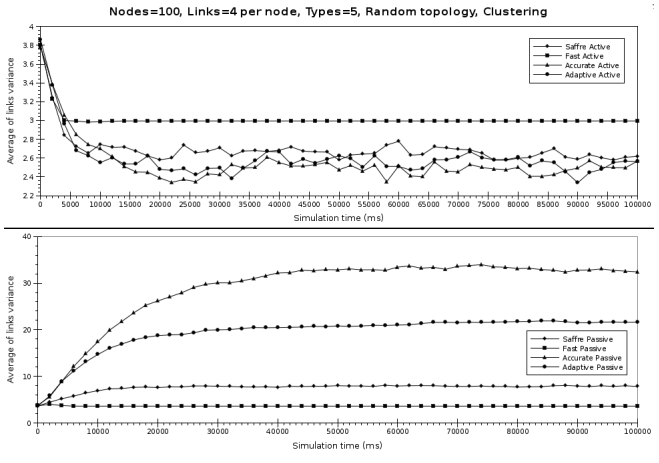

Figure 8: Reference Experiment: links variance.

4.2.2 Varying the Number of Types

In Table 1 we can see that if we reduce the number of types all the algorithms perform better than if increasing it. A first explanation is that the initial optimality tends to be $opt_{init} \approx \frac{1}{N_{types}}$ where N_{types} is the number of types, therefore if the initial optimality is lower, more iterations are needed to reach the global optimum. The second explanation is that if N_{types} is lower, then the probability to find equal neighbors after each iteration is higher because there are less different nodes. The fact that the algorithm requires a higher number of iterations to reach convergence when increasing the types results in a slight increase in the number of messages.

Table 1: Comparison of the algorithms optimalities after 2s and after 100s when changing the number of types to the reference experiment.

Types:	**2**	**5**	**10**	**15**
Initial. opt.	50%	20%	9.2%	5.7%
Algorithm	**2s/100s**	**2s/100s**	**2s/100s**	**2s/100s**
Act. Saffre	66%/85%	38%/57%	23%/35%	17%/25%
Act. Fast	70%/84%	38%/55%	24%/34%	17%/24%
Act. Accurate	67%/98%	26%/94%	14%/84%	8.6%/71%
Act. Adaptive	66% 94%	36%/87%	21%/82%	15%/77%
Pass. Saffre	65%/86%	39%/58%	27%/36%	20%/26%
Pass. Fast	73%/86%	44%/57%	28%/34%	21%/24%
Pass. Accurate	59%/88%	26%/65%	12%/42%	8.6%/32%
Pass. Adaptive	71%/96%	40%/82%	26%/62%	19%/48%

4.2.3 Varying the Number of Nodes

This test has been performed to see how much scalable are the algorithms when changing the number of nodes. From Figure 6 we can see that this does not affect the optimality with respect to the experiment of reference in most of the cases. The slight reduction in the local convergence (optimality after 2s) is due to the fact that the network is slowed down by the additional messages sent by the new nodes.

Table 2: Comparison of the algorithms optimalities after 2s and after 100s when changing the number of nodes to the reference experiment.

Nodes:	100	200	300
Initial. opt.	20%	20%	20%
Algorithm	**2s/100s**	**2s/100s**	**2s/100s**
Act. Saffre	38%/57%	35%/57%	32%/56%
Act. Fast	38%/55%	39%/56%	40%/57%
Act. Accurate	26%/94%	26%/94%	21%/86%
Act. Adaptive	36%/87%	33%/87%	24%/83%
Pass. Saffre	39%/58%	36%/57%	32%/57%
Pass. Fast	44%/57%	41%/57%	36%/57%
Pass. Accurate	26%/65%	25%/64%	23%/66%
Pass. Adaptive	40%/82%	34%/85%	29%/89%

4.2.4 Varying the Number of Links

A modification in the number of links has the following effect: it makes node communication easier because it increases the probability that a given neighbor has a connection to a node of the required type. Thanks to this effect the highly-constrained algorithms (Saffre and Fast algorithms) can perform more link exchanges and increase the optimality. On the other hand having more links may generate additional noise for the other algorithms that can slow down their convergence rate. This explains why when increasing the number of links per node the Saffre and the Fast algorithms become better while Accurate and Adaptive become worse (these results can be found in Table 3).

Table 3: Comparison of the algorithms optimalities after 2s and after 100s when changing the number of links to the reference experiment.

Links per node:	3	4	5
Initial. opt.	19%	20%	19%
Algorithm	**2s/100s**	**2s/100s**	**2s/100s**
Act. Saffre	39%/50%	38%/57%	38%/62%
Act. Fast	42%/51%	38%/55%	39%/59%
Act. Accurate	31%/94%	26%/94%	26%/94%
Act. Adaptive	38%/88%	36%/87%	36%/86%
Pass. Saffre	40%/51%	39%/58%	37%/63%
Pass. Fast	43%/51%	44%/57%	44%/60%
Pass. Accurate	28%/69%	26%/65%	26%/62%
Pass. Adaptive	42%/85%	40%/82%	40%/80%

4.2.5 Varying the Initial Topology

In this experiment we want to see if the results we have obtained till now are generalizable if we change the way in which nodes are connected at start-up. From Table 4 we can see that the *torus topology* makes all algorithms go slower (and therefore with more messages) because the average distance between two nodes is larger (expressed in terms of number of links to traverse). A larger distance between nodes requires therefore more iterations and links exchanges in order to connect distant nodes. The opposite of this phenomenon can be seen in the *scale-free* experiments, in which the distance between nodes is small. Apparently strange results can be observed with the *star* experiments in which most of the nodes have only one neighbor and therefore in many situations they are not able to start an algorithm iteration.

Table 4: Comparison of the algorithms optimalities after 2s and after 100s when changing the initial topology to the reference experiment.

Topology:	Random	Torus	Scale-free	Star
Initial. opt.	20%	4.0%	19%	20%
Algorithm	**2s/100s**	**2s/100s**	**2s/100s**	**2s/100s**
Act. Saffre	38%/57%	7.1%/11%	34%/68%	28%/97%
Act. Fast	38%/55%	7.4%/8.1%	38%/64%	29%/96%
Act. Accurate	26%/94%	7.6%/93%	24%/92%	22%/90%
Act. Adaptive	36%/87%	11%/86%	32%/88%	27%/97%
Pass. Saffre	39%/58%	8.8%/11%	33%/70%	21%/73%
Pass. Fast	44%/57%	8.1%/8.2%	35%/67%	21%/94%
Pass. Accurate	26%/65%	6.8%/65%	25%/59%	19%/34%
Pass. Adaptive	40%/82%	13%/67%	33%/73%	21%/82%

4.2.6 Limiting the Maximum Number of Neighbors

This test has been performed to see what happens to the passive algorithms when limiting the maximum number of neighbors for a node. The limit has been introduced to make the links variance as low as in the active algorithms in order to avoid the creation of the scale-free topologies we have seen in Figure 8. Looking at Table 5 we can see that if this limit is not too low (for example 8) we have an optimality improvement with respect to the reference experiment. This can be explained by the fact that the "noisy" iterations responsible for the creation of the scale-free topology are removed, therefore the algorithm can converge in a smoother way. However, if the limit is too low (for example 4) we obtain the opposite effect on the optimality because many nodes become stuck due to the too restrictive limit. Finally, the reduction of the links variance, particularly for the noisy algorithms (Accurate and Adaptive), can be seen in Table 6.

Table 5: Comparison of the algorithms optimalities after 2s and after 100s when changing the neighbors limit to the reference experiment.

Max Neighbors:	4	6	8	unlimited
Initial. opt.	19%	18%	19%	20%
Algorithm	**2s/100s**	**2s/100s**	**2s/100s**	**2s/100s**
Pass. Saffre	27%/39%	33%/56%	36%/60%	39%/58%
Pass. Fast	26%/37%	37/53%	41%/55%	44%/57%
Pass. Accurate	20%/42%	23%/78%	25%/78%	26%/65%
Pass. Adaptive	27%/48%	33%/82%	37%/87%	40%/82%

Table 6: Comparison of the algorithms links variances after 2s and after 100s when changing the neighbors limit to the reference experiment.

Max Neighbors:	4	6	8	unlimited
Initial. linksv.	3.8	3.4	3.9	3.7
Algorithm	**2s/100s**	**2s/100s**	**2s/100s**	**2s/100s**
Pass. Saffre	3.2/1.7	3.2/2.1	4.2/4.2	4.5/7.8
Pass. Fast	3.3/2.5	3.3/2.9	3.7/3.3	4.0/3.6
Pass. Accurate	3.5/0.52	4.0/3.7	5.0/7.6	5.6/32
Pass. Adaptive	3.2/1.6	3.8/4.0	5.0/7.7	5.8/22

4.3 Summary

In these experiments we have compared four clustering algorithms executed with different modes. What we have learned is that, if we need speed, the fast and the adaptive algorithms are good choices in almost all situations. If we want to reduce the number of messages, the fast algorithm with passive protocol (with a limit in the number of neighbors if super-nodes are not acceptable) is usually the best choice. If homogeneity or optimality are critical factors, then the accurate and the adaptive algorithms are preferable. In conclusion, the adaptive algorithm, due to its "adaptive" nature, performs well in most common situations, however in extreme situations it can be overcome by the others.

Some experiments have been executed by using the reverse-clustering algorithm: the results are pretty similar, with the difference that the increase in the number of types in this case improves the algorithm results instead of the opposite.

5. RELATED WORK

Self-aggregation techniques are based on the principles of the cluster analysis that seeks to identify homogeneous subgroups of cases in a population. This is a widely known discipline applied in areas like economics, social sciences and also in software engineering [8]. Cluster analysis aims at identifying a set of groups which both minimize within-group variation and maximize between-group variation.

However, these techniques are mainly applied using a centralized approach, where a dedicated entity is in charge of

establishing the desired global property applying suitable techniques/algorithms. The paradigm of self-aggregation, instead, is to distribute the responsibilities among the individual entities: no single entity is in charge of the overall aggregation, but each contributes to a collective behavior. Following this philosophy, mainly inherited from natural adaptive systems, the local behavior rules applied in all entities lead (hopefully) to the desired global behavior. Examples of application of these rules can be found in the area of communication networks: for example for the control of topology in wireless multi-hop network [7], or the computation of a maximal independent set in radio networks [18]. Several kind of application of self-organization techniques in communication network are reviewed in [20].

Apart from self-aggregation approaches, bio-inspired techniques have recently be applied in several fields, spanning the robot self-organization [25], the behavior of autonomic network [4], the actions of swarms of autonomous vehicles performing dangerous tasks [26], and to organize the sensor network deployment [12].

Different lines of research apply methods based on the use of genetic algorithms or neural networks to define and to study the problems related to cluster formation, e.g., [2] or the multi-agent approach like in [22].

In this paper we have tried to understand the rationale of the behavior of existing self-aggregation techniques working on a distributed environment and giving rise to collective behaviors on a large scale. Besides, starting from these results we have defined some improved versions of the algorithms that upgrade the self-aggregation performances in specific cases.

6. CONCLUSIONS

In this paper we have proposed some self-aggregation algorithms that, based on simple local rules, are able to determine some global properties to the whole system without any centralized control nor scaling issues.

The study on these algorithm has been performed by simulating their execution through a distributed simulator. The results of this analysis have been used to identify strengths and weaknesses of each algorithm, and therefore to produce self-decision heuristics that can be used to choose the algorithm that best fits a particular situation.

Acknowledgments

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

7. REFERENCES

[1] Self-Aggregation Simulation Results. `http://home.dei.polimi.it/dinitto/papers/self-data.zip`.

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

[3] O. Babaoglu, H. Meling, and A. Montressor. Anthill: A framework for the development of agent-based peer-to-peer systems. *Proc. Int. Conf. Distributed Computer Systems*, 2002.

[4] S. Balasubramaniam and al. Bio-inspired policy based management (biopbm) for autonomic bio-inspired policy based management (biopbm) for autonomic. In *POLICY*, pages 3–12. IEEE Computer Society, 2006.

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

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

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

[8] E. Brian, S. Landau, and M. Leese. *Cluster analysis*. Edward Arnold Publishers Ltd, 2001.

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

[10] J. Deneubourg and al. Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical Biology*, pages 105, 259–271, 1983.

[11] D. Devescovi, E. Di Nitto, D. J. Dubois, and R. Mirandola. Self-Organization Algorithms for Autonomic Systems in the SelfLet Approach. In *International Conference on Autonomic Computing and Communication Systems, Autonomics 2007*. ACM, 2007.

[12] S. Dolev and N. Tzachar. Empire of colonies: Self-stabilizing and self-organizing distributed algorithms. In *Principles of Distributed Systems, 10th International Conference, OPODIS 2006*, volume 4305 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 2006.

[13] D. J. Dubois, R. Mirandola, and E. Di Nitto. Work Package 3 Deliverable 3.3: Software Implementation of Modules for Adaptive Aggregation. `http://www.cascadas-project.com`.

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

[15] IBM. Autonomic vision and manifesto. Website. URL: `http://www.research.ibm.com/autonomic/manifesto/`.

[16] P. Marrow and al. Work Package 3 Deliverable 3.1: Aggregation Algorithms, Overlay Dynamics and Implications for Self-Organized Distributed Systsems. http://www.cascadas-project.com.

[17] A. Montresor, M. Heing, and O. Babaoglu. Messor: Load-balancing through a swarm of autonomous agents. In *Proceedings of the 1st International Workshop on Agents and Peer-to-Peer Computing*, Bologna, Italy, July 2002.

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

[19] S. Nicolis and al. Optimality of collective choices: a stochastic approach. *Bulletin of Mathematical Biology*, pages 65, 795–808, 2003.

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

[21] F. Saffre and R. Ghanea-Hercock. Simple laws for complex networks. *BT Technology Journal*, 21(2), 2003.

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

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

[24] M. Shackleton, F. Saffre, R. Tateson, E. Bonsma, and C. Roadknight. Autonomic computing for pervasive ict - a whole-system perspective. *BT Technology Journal*, 22(3), 2004.

[25] W.-M. Shen, B. Salemi, and P. Will. Hormone-inspired adaptive communication and distributed control for conro self-configurable robots. *IEEE Transaction on Robotics and Automation*, 2002.

[26] W. Xi, X. Tan, and J. S. Baras. A stochastic algorithm for self-organization of autonomous swarms. In *Proceedings of the 44th IEEE conference on Decision and control and the european control conference*. IEEE Computer Society, 2005.