An Evaporative Approach to Handle Dynamics in Diffusive Aggregation Schemes

Nicola Bicocchi
Dipartimento di Scienze e Metodi dell'Ingegneria
Universita' di Modena e Reggio Emilia
nicola.bicocchi@unimore.it

Marco Mamei
Dipartimento di Scienze e Metodi dell'Ingegneria
Universita' di Modena e Reggio Emilia
marco.mamei@unimore.it

Franco Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria
Universita' di Modena e Reggio Emilia
franco.zambonelli@unimore.it

ABSTRACT

Distributed computing in large-size dynamic networks often requires the availability at each and every node globally aggregated information about some overall properties of the network. In this context, since traditional broadcasting solutions become inadequate as the number of participating nodes increases, aggregation schemes inspired by the physical/biological phenomenon of diffusion have been recently proposed as a simple yet effective alternative to solve the problem. However, diffusive aggregation requires specific solutions to cope with the dynamics of the network and/or of the values being aggregated. While typical solutions are based on periodic restarts (epoch-based approaches), in this paper, we propose an original and more autonomic solution, relying on coupling diffusive aggregation schemes with the additional bio-inspired mechanism of evaporation. While a gossip-based diffusive communication scheme is used to aggregate values over a network, gradual evaporation of values can be exploited to account for network and value dynamics without requiring periodic restarts. A comparative performance evaluation shows that the evaporative approach is able to manage the dynamism of the values sensed over the network in an effective way and, in the most of the cases, it leads to more accurate aggregate estimations than epoch-based techniques.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Systems]: Distributed Systems

General Terms: Algorithms, Design, Performance

Keywords: Large Networks, Diffusive Aggregation, Sensor Networks

1. INTRODUCTION

Controlling and orchestrating global distributed computations in large-size dynamic networks, such as sensor networks [7], P2P overlays and grids [6, 9, 1], is particularly challenging. In fact, one has to account for a large number of nodes dynamically joining and leaving the network and changing their local states.

A recurrent distributed computing problem for these kinds of networks is that of making available at each and every node information about some global properties of the network (e.g., the average load, the current number of nodes) or of the environment in which the network is deployed (e.g. the average temperature measured by a wireless sensor network). Many techniques have been developed in the recent past to produce such information (e.g., techniques based on flooding [14] or on spanning trees [15]). However, these appear suitable only for networks of limited size and dynamism. To deal with large-size networks, other directions have to be, and have been, explored.

Protocols inspired by epidemic metaphors are very promising in such a scenario (see [8] for an excellent introduction to the area). In particular, schemes inspired by the physical/biological phenomenon of *diffusion* have been proposed as a simple yet effective solution to propagate local values over a large-size network and to aggregate values over the whole network [11, 2], or a portion of it [3]. Several applications of diffusive-based aggregation schemes have been already reported in the literature [2, 4, 16, 5, 13].

In diffusive-based aggregation schemes, each node in a network can locally obtain a global picture of some properties of the network, and can exploit such knowledge to overpass its local viewpoint and adapt/control its activities in a more informed way. However, in dynamic situations, the global representation available at a node is at risk of ageing, and of no longer reflecting the current actual global state of the network. Indeed, aggregation algorithms, to be effective, should be able to manage situations in which *(i)* nodes dynamically join and leave the network and *(ii)* the values that are being aggregated change over time.

The most widely adopted technique to deal with dynamic situations is the so-called epoch-based one, relying on periodic restarting of the aggregation process [11]. Periodically, aggregated values (which may no longer reflect the current situation) are invalidated and substituted with those resulting by the execution of a new aggregation process (i.e., of a new "epoch"). Unfortunately, as we will discuss in this paper, epoch-based approaches can induce to highly inaccurate and unreliable results in aggregation, other than requiring a careful tuning of internal parameters.

The contribution of this paper is to propose an innovative technique to handle dynamics in diffusive-based aggregation protocol. The inspiration behind our idea is that of

BADS'09, June 19, 2009, Barcelona, Spain.

pheromone evaporation [12, 10], which makes the process of ant-foraging capable of adaptively deal with dynamic environment. Pheromone trails that are no longer valid slowly evaporate, while those that are still useful get continuously re-enforced. Analogously, our idea – which totally avoids periodic restarts – is that of making aggregated values slowly evaporate. In this way, aged aggregated values that do no longer reflect an up-to-date situation disappear, while those that are still up-to-date will be re-enforced.

Experimental results performed on a wide range of situations, together with comparative evaluations against two variants of the epoch-based technique, show that the evaporative approach is able to handle the dynamism of the values measured over the network in an effective way, and in the most of the cases is more accurate than epoch-based techniques. In particular, the evaporative solution is to be preferred either in very-large networks or whenever periodic restarts are to be avoided.

The rest of this paper is organized as follows. Section 2 introduces diffusive aggregation protocols and discusses dynamic situations. Section 3 describes two variants of the epoch-based technique based on periodic restarts, and outlines their limitations. Section 5 introduces and details our novel evaporative approach. Sections 6 evaluates, also in a comparative way, epoch-based approaches and our evaporative approach. Section 7 concludes.

2. DIFFUSIVE AGGREGATION

In this section we introduce the basic algorithm and gossip-based communication scheme that are typically adopted to implement diffusive-based aggregation in large-size networks, and the typical problems which arise in the presence of dynamic situations.

2.1 The Basic Algorithm

The three approaches we are going to compare in this paper share the same diffusive aggregation algorithms and rely on the same gossip-based communication scheme.

Let us consider a network of n nodes in which each of the nodes is connected with a small ($<< n$) number of other nodes (neighbors). The gossip-based communication protocol is based on the "push-pull" scheme illustrated in Figure 1. Time is considered divided into slices of length t. Each node p executes two different threads. At every slice, the active thread starts an information exchange with a random subset of its neighbors q by sending them a message containing the local state s_p and waiting for a response with the remote state s_q. The passive thread waits for messages sent by an initiator and replies with the local state. The term $push - pull$ refers to the fact that each information exchange is performed in a symmetric manner: both participants send and receive their states. This is a general scheme and the local state s_p could represent whatever property of the node p or whatever value measured by it.

The method GETNEIGHBORS(ω) can be thought of as an underlying service to the gossip-based protocol, which is normally (but not necessarily) implemented by sampling a locally available set of neighbors. We assume that GETNEIGHBORS(ω) returns a uniform random sample over the entire set of neighbors. Moreover the parameter ω can be used to specify how many neighbors have to be returned. In particular, with $\omega = 1$ it will return all the available neighboring nodes while with $\omega = 1/neighbors$ only one neighbor will be returned (as in [11]). It is convenient to describe the GETNEIGHBORS(ω) method as a separate service in order to hide the physical differences between networks. For example over an IP network, neighbors are nodes connected to the same collision domain while in a WSN, neighbors are nodes with a relative physical distance lower than their radio range.

Active Thread

do exactly once in each consecutive t time units at randomly picked time:
    $q[] \leftarrow$ GETNEIGHBORS(ω)
    foreach q:
        send s_p to q
        $s_q \leftarrow$ receive(q)
        $s_p \leftarrow$ UPDATE(s_p, s_q)

Passive Thread

do forever:
    $s_q \leftarrow$ receive($*$)
    send s_p to sender(s_q)
    $s_p \leftarrow$ UPDATE(s_p, s_q)

Figure 1: The basic gossip-based protocol for diffusive aggregation rely on a "push-pull" scheme. Time is considered divided into slices of length t. Each node p executes two different threads. At every slice, the active thread starts an information exchange with a random subset of its neighbors q by sending them a message containing the local state s_p and waiting for a response with the remote state s_q. The passive thread waits for messages sent by an initiator and replies with the local state.

The method UPDATE computes a new local state based on the current local state and the remote state received during the information exchange. It is within the UPDATE method, in particular, that the diffusive aggregation process takes place. The output of UPDATE and the semantics of the node state depend on the specific aggregation function being implemented by the protocol. Figure 2 reports three exemplary instances of the UPDATE method for the minimum, maximum, and average aggregation functions.

This gossip-based communication scheme on which diffusive aggregation relies acts as a sort of continuous background activity. It tends to impose a predefined, tunable, and uniform load, to the system. Each node executes the same amount of operations. Incidentally, shorter t or higher ω speed up the convergence of the algorithm and increase the number of exchanged messages. Therefore, every user can select the most suitable trade off between accuracy and communication costs over time. Incidentally on networks with bounded energy capacity like WSN communication costs take a great relevance because of they are strictly related with the lifespan of the system. Thus, being able to tune communication costs could be useful in practical settings.

UPDATEMIN(s_p, s_q):
return $min(s_p, s_q)$

UPDATEMAX(s_p, s_q):
return $max(s_p, s_q)$

UPDATEAVG(s_p, s_q):
return $(s_p + s_q)/2$

Figure 2: The UPDATE functions for three different aggregation functions (i.e. minimum, maximum, average).

2.2 Handling Dynamics

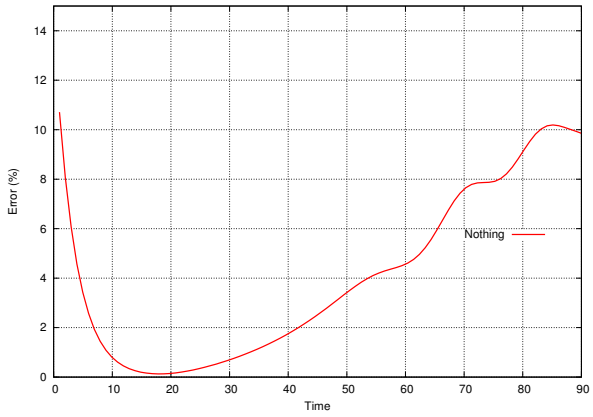

Figure 3: The average error rate of a diffusive aggregation scheme without any mechanism to handle environment dynamics (up), compared with the one produced by our original evaporation-based approach (EVP) and two epoch-based approaches (GEA, GEAopt).

The generic protocol described so far is not adaptive, in that aggregation does not take into account neither the dynamism of the network nor the changes in the values being aggregated.

Let us assume that each node in the network holds a numeric value, which can characterize some aspects of the node or its environment (for example, the load at the node, storage space, atmospheric pressure). The goal of a diffusive aggregation is to continuously provide the nodes with an up-to-date estimate of an aggregate function, computed over the values held by the current set of nodes. However, in a realistic scenario, nodes continuously join and leave the network. Moreover values measured by the nodes may change over time. From this point of view, changes of the values held by the nodes or changes of the network itself (e.g. nodes leaving and joining) could be considered the same. For example a node that suddenly starts measuring a *null* value can be considered as a node that left the network.

In Figure 3 we report the average error on the aggregated data we obtained during the execution of a diffusive aggregation scheme (average aggregation of a scalar value) and in the absence of any mechanism to handle dynamics. Starting from an initial state in which the nodes do not yet have any accurate estimation of the global value, the error quickly decreases as the algorithm converges and the nodes acquire a correct local estimation of the aggregated value. Nevertheless, after some iterations and while the local situation at nodes changes because of some dynamic phenomenon (e.g., the values at some nodes diminish), the average error starts to increase (i.e., the aggregated value does no longer reflect reality) and goes out of any control. This happens because there is no mechanism to keep aggregated data updated.

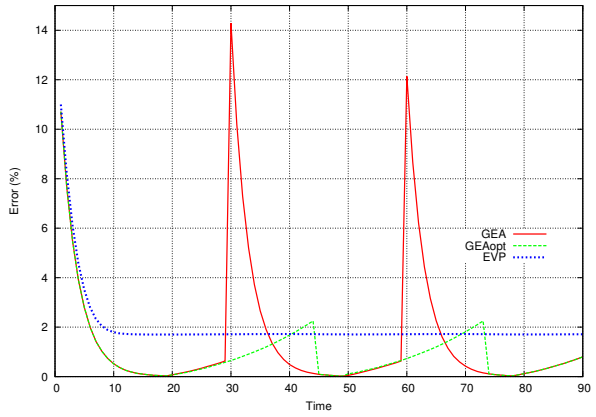

Figure 4: The average error rate of a diffusive aggregation scheme without any mechanism to handle environment dynamics.

It can be shown (and it is quite intuitive indeed, due to the cumulative nature of aggregation) that aggregation algorithms do not experience problems if executed on a fixed or growing number of nodes measuring steady phenomena. Instead, the case in which the network shrinks, and the case in which the values to be aggregated change are more complex to be managed. In fact, in both of these situations the aggregated values may no longer be valid and the cumulative nature of aggregation does not enable them to be properly updated. For example if a node holding a very high value leaves the network, several aggregated variables (e.g. maximum or average) propagated to every node, may no longer reflect the current picture of the network.

Considering that all natural and social processes involve changes, and that most modern networks are of an inherent dynamic nature, techniques are required to manage this recurrent situation.

3. EPOCH-BASED APPROACH

To solve the problem and make aggregation schemes able to handle dynamics, the simplest solution is implement periodic restarts of the schemes [11] , i.e., realizing a so-called epoch-based aggregation scheme.

3.1 Basic Algorithm

Starting from the algorithm proposed in Figure 1, we present here the simplest possible form of epoch-based diffu-

sive aggregation. In the following of the paper, we will refer to it as GEA (Gossip-based Epoch Aggregator).

To implement termination, we adopt a very simple mechanism: each node executes the protocol for a predefined number of cycles, denoted as γ , depending on the required accuracy of the output and the convergence time of the aggregation process. To implement restarts, we divide the protocol execution into consecutive epochs of length $\delta = \gamma t$ (where t is the cycle length), and start a new instance of the protocol at each epoch. We also assume that messages are tagged with an epoch identifier. To avoid synchronization problems, if a node receives a new epoch identifier, it immediately joins the new epoch.

Active Thread

$epochTime = 0$
do exactly once in each consecutive
t time units at randomly picked time:
    $q[] \leftarrow$ GETNEIGHBOR(ω)
    foreach q:
        send s_p to q
        $s_q \leftarrow$ receive(q)
        $s_p \leftarrow$ UPDATE(s_p, s_q)
    if $epochTime >= \gamma t$:
        RESET(s_p)

Figure 5: GEA algorithm main body.

Figure 5 shows the basic gossip-based communication scheme presented in the previous section, yet modified with the introduction of the *epochTime* variable to divide the execution into different time slices. For each execution, aggregated values are cleared and recomputed again.

Figure4 shows the trend of the average error in the time domain during the execution of an epoch-based aggregation scheme. This highlights an important drawback of GEA which is independent of the chosen *epochTime*. Every time an epoch ends, local aggregated values are set to the node's local value. As a consequence of this, the average error suddenly increases up and a portion of the next epoch is needed to make the algorithm converge again. It is clear that the low stability of the error rate is not a desiderable property and, under several circumstances, could be unacceptable.

3.2 Optimized Algorithm

As introduced before, Figure 4 shows the biggest drawback of GEA. Every time an epoch ends, local aggregated values are set to the nodes' local values. Due to this, the average error suddenly steps up and a variable portion of the following epoch is consumed to lower the error. In order to guarantee a minimum quality of service, it is important to avoid these spikes.

Due to this, a modified version of GEA is usually adopted, we hereby refer to this as GEAopt, in which nodes are able to participate to two epochs simultaneously. By this mean: *(i)* nodes are able to answer queries using the data aggregated over an already properly converged epoch; *(ii)* at the same time, they are computing new aggregated data over a new concurrently executing epoch. Once the new epoch of aggregation is assumed to have properly aggregated updated values, the data of the older aggregation can be dismissed, nodes can start answering queries with the newly computed aggregated data, and a further concurrent epoch can be launched to start computing the next-new aggregated values. It is worth emphasizing that the communication costs are not higher than in the GEA approach because different epochs are computed in parallel using the same data exchanged in GEA . This mechanism enforced in GEAopt can greatly reduce error spikes and thus produces better performances than GEA. However, as it can be seen from from Figure 4, the issue of having a great variability in the errors produced still exists. In fact, while the new epoch is being aggregating new data, the old epoch reports on aggregated data which, in the presence of dynamics, tends to be increasingly inaccurate. Also, the GEAopt approach introduces the issue of evaluating the time in which the old epoch should be dismissed and the new one should step up. In our experiment, we choose a static policy for our experiments in which GEAopt replies with values aggregated over the new epoch when it has accomplished more than half of its length.

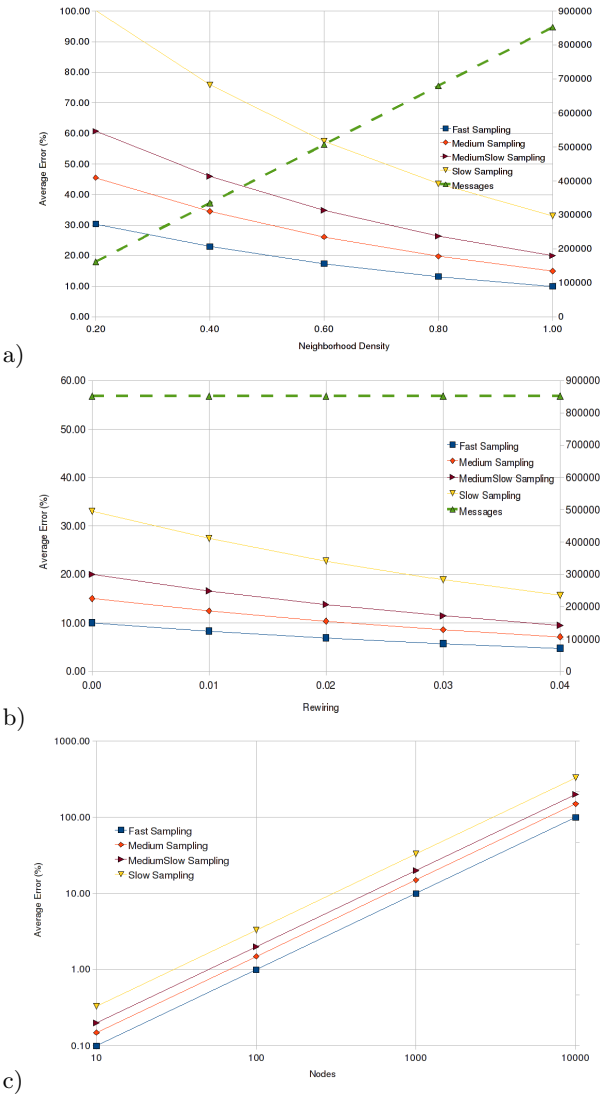

Figure 6: GEA performances varying network (a) density, (b) rewiring and (c) size.

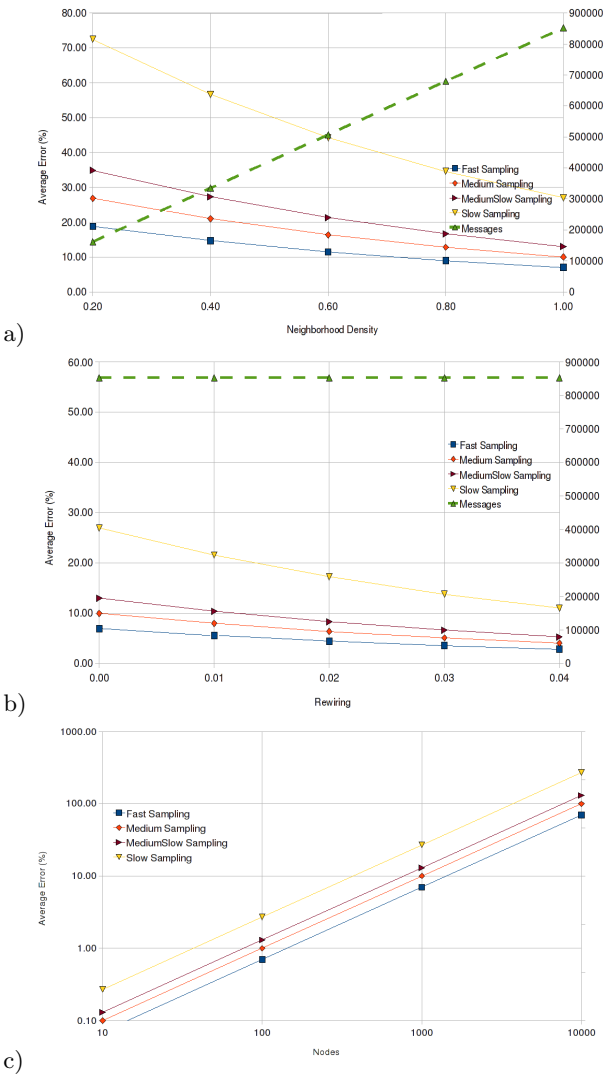

Figure 7: GEAopt performances varying network (a) density, (b) rewiring and (c) size.

4. THE EVAPORATIVE APPROACH

The two above proposed aggregation schemes share the need of periodic restarts. This keeps them fairly simple and efficient. However, in several circumstances, a continuous approach which never needs to be restarted can be useful. For example, over very large networks where the convergence time could be high, an approach able to avoid periodic restarts and, thus, error spikes, should be preferred.

From a conceptual viewpoint, we can think at further improving GEAopt with multiple concurrent epochs instead of only two. In this way, at any given time, one can be sure that an epoch whose values are close to convergence does exist. Yet, this leaves open the issue of deciding which is the best one. We would like to overcome this limitation and propose a mechanism able to mimic the results that would be achievable with a very high, almost infinite, number of parallel epochs avoiding the need of selecting the best one and, in the end, periodic restarts at all.

The basic principle behind our idea is based on the mechanism used by ants to reach food's sources. In particular they are able to select a good path to the food by reinforcing pheromone's trails which are constantly evaporating. In the same way, our approach "evaporates" the values being aggregated, leaving the past ones to disappear and reinforcing the up to date ones. In other words, the aggregated values at a node are slowly (compared to the convergence time of the aggregation algorithms) moved towards the local values of the node. In this way, the weight of those data cumulated by the algorithm will gradually diminish, unless properly re-enforced. The result (as it is clear from Figure 4) is that the estimate error of an aggregated value is much less variable, slightly oscillating around a small value, and never exhibiting the spikes typical of epoch-based approaches.

This new approach maintains the general diffusion scheme presented in Figure 1 but exploits a particular logic into the function UPDATE.

UPDATEMIN(s_p, s_q):
$m_p = min(s_p, s_q)$
return $min(m_p + \Delta, v_p)$

UPDATEMAX(s_p, s_q):
$M_p = max(s_p, s_q)$
return $max(M_p - \Delta, v_p)$

UPDATEAVG(s_p, s_q):
$w_p = \frac{w_p + w_q + v_q - v_p}{2}$
if $(abs(w_p) < \Delta) w_p = 0$
if $(w_p < -\Delta) w_p = w_p + \Delta$
if $(w_p > \Delta) w_p = w_p - \Delta$
return w_p

Figure 8: The UPDATE* functions of the evaporative approach, as used to compute the aggregated values of *minimum*, *maximum* and *average*.

As an example, consider the case of a network aggregating the maximum of a local property. Assume that each node has already locally available its own maximum estimator. Now, have each node slightly "evaporates" such value by making it diminishes to approach the local value. If the node holding the actual maximum doesn't change, a node will receive again the maximum value undoing the evaporation effects. Instead, if the value measured by the node holding the maximum decreases, evaporation will stabilize the new maximum at each node, after proper evaporation. Similar considerations can be applied to all the other aggregation functions [1].

Another very relevant aggregation function, worth exemplifying, is the average. The algorithm we propose for calculating the average in a fully decentralized way apparently only slightly differs from that discussed in [11]. However, it has the advantage of making it possible to tolerate the evaporative approach. The algorithm works as follows: let p and q be two neighbor sensors and $v_p(t)$ and $v_q(t)$ be the values sensed by s_p and s_q, respectively at time t. Then define the average estimator avg as:

[1]More precisely this approach applies to the class of order and duplicate insensitive aggregated functions [14]

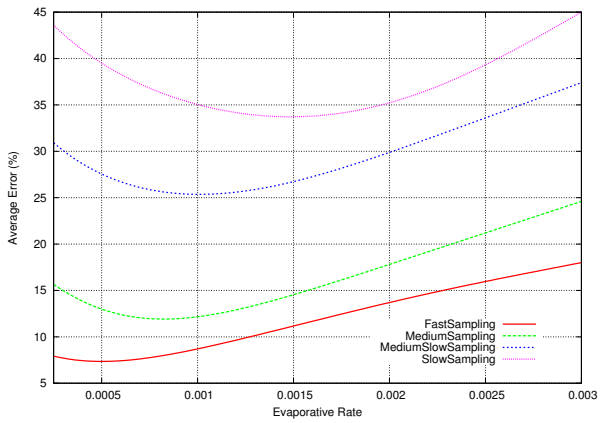

Figure 9: The effects of the evaporative rate (Δ) over the EVP approach. For each possible value of t we run several simulations using different values of Δ ($n = 1000, \alpha = 0$, $\omega = 1$). It is clear from the graph that for every t exists an optimum value of Δ which produces the lowest error. Moreover it is clear that to compensate low sampling frequencies higher values of Δ are required.

$$avg_p = v_p(t) + w_p(t); w_p(0) = 0$$
$$avg_q = v_q(t) + w_q(t); w_p(0) = 0$$

We define the average estimator as the sum of the local value measured by the node and a second value w which represents the difference between the local value and the estimated average. w is the term subject to evaporation. Decoupling w from v allows the evaporation process: positive values of w has to be diminished, negative increased. At every iteration we aim to update the value of w to let each node converge to the right value of avg. In particular:

$$\begin{cases} w_p(t) + w_q(t) = w_p(t+1) + w_q(t+1) \\ v_p(t) + w_p(t+1) = v_q(t) + w_q(t+1) \end{cases}$$

Resolving in $w_p(t+1)$ and $w_q(t+1)$ it leads to:

$$\begin{cases} w_p(t+1) = (w_p(t) + w_q(t) + v_q(t) - v_p(t))/2 \\ w_q(t+1) = (w_p(t) + w_q(t) + v_p(t) - v_q(t))/2 \end{cases}$$

It can be shown that by asynchronously applying the above calculation for subsequent couples of nodes in the system, whatever the order is, makes the average estimator avg at each node converge toward the global average of the value v in the region. And that the convergence is not undermined by dynamic changes in the values v of the nodes of the network.

In Figure 8, we reported three different UPDATE functions to manage *minimum*, *maximum* and *average* estimation using our evaporative technique (EVP).

Clearly EVP performances are also influenced by the evaporative rate Δ (the only parameter of this approach). For each possible value of t we run several simulation using different values of Δ.

Figure 9 shows the relation between the average error rate and Δ. It is clear from the graph that for every t, an optimum value of Δ which produces the lowest error exists. Moreover it is clear how, to compensate low sampling frequencies higher values of Δ are required. This happens because a bigger time slot between two consecutive samplings allows bigger changes into the environment. The more the values' fluctuations increase, the more Δ should be high to follow them. In particular this property could be the starting point to apply the evaporative mechanism to systems which have to deliver an autonomic behavior by self adapting Δ. However, it is also interesting to note that the performances of EVP are not that greatly dependent over the value of Δ chosen. In fact, even if EVP is not able to self-adapt the values of Δ, changes in its value don't produce much greater errors over the optimal value.

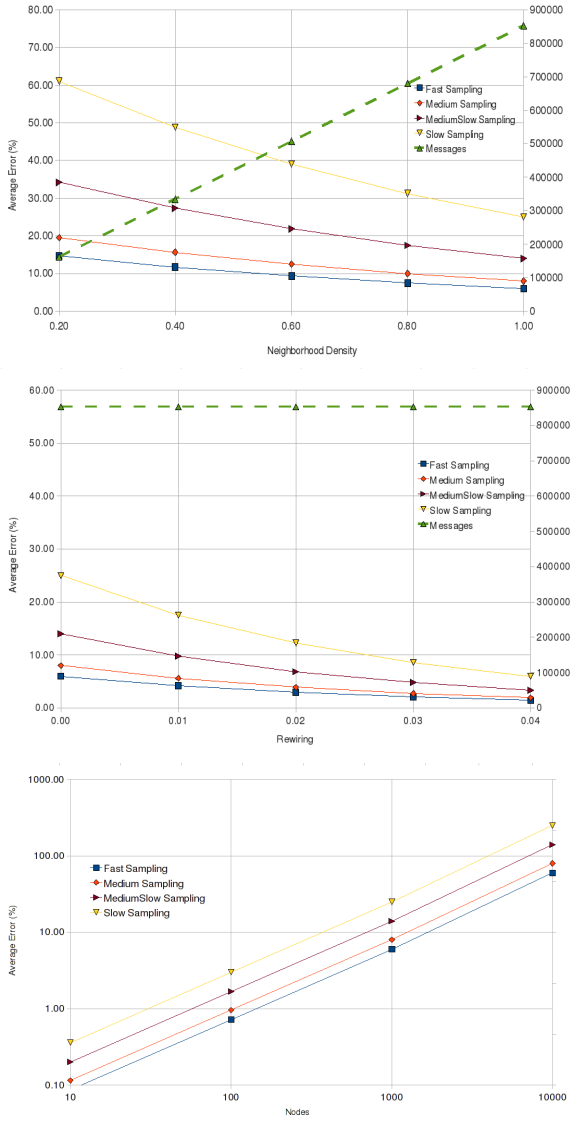

Figure 10: EVP performances varying network (a) density, (b) rewiring and (c) size.

5. PERFORMANCE EVALUATION

5.1 Experiments Setup

To measure the performances of the aforementioned approaches under different conditions we performed several experiments varying the relevant parameters. We chose to study the effects of four parameters: two related to the algorithm itself and two related to the network over which the algorithm is executed. In particular we evaluated the sampling period (t), the neighborhood density (ω), the number of nodes of the network (n) and the rewiring percent of the network (α) measuring the randomness of network links as detailed later.

Concerning t, we were interested in the effects of the speed of the observed phenomenon over the aggregation performances. In particular, given a phenomenon which needs a known sampling period T to be properly monitored, we conducted experiments using: $t = T/3$ ($fastsampling$), $t = T(mediumsampling)$, $t = 2T$ ($mediumslowsampling$), $t = 3T(slowsampling)$. By this mean, from the algorithm viewpoint, $fast$ sampling implies the perception of a slow phenomenon, $slow$ sampling implies the perception of a fast one and so on.

We evaluated also the behavior of the algorithms varying ω in the range between 0.2 and 1.0. Given our experiments' setup, this range is enough to understand the general effects of this parameter.

Concerning the network, we examined up to now only static topologies (neighbors don't change in time). While static topologies are unrealistic in the presence of churn, we still consider them due to their theoretical importance and because of they represent a fair model of several network setups. We executed all the simulations starting from a 4-neighbors regular lattice network. Then, increasing the link's rewiring probability (α), we studied the influences of emerging random network effects on the proposed approaches. The lattice is built by connecting the nodes to their nearest neighbor until the desired node degree is reached. Starting with this, each edge is then randomly rewired with probability α. Rewiring an edge at node p means removing that edge and adding a new edge connecting p to another node picked at random. When $\alpha = 0$, the lattice remains unchanged, while when $\alpha = 1$, all edges are rewired, generating a random graph. For intermediate values of α, the structure of the graph lies between these two extreme cases: complete order and complete disorder. Considering that with $\alpha > 0.05$ the network fundamental properties don't change significantly, we reported only the results obtained with $\alpha < 0.05$.

We also evaluated the scalability of the algorithms varying the size of the 4-neighbors regular lattice from 10 up to 10000 nodes.

Every experiment has been conducted over a simulation of 250 iterations. Every point in Figures 6, 7, 10 represents results averaged over 10 simulation runs. In all the experiments we fixed all the parameters except the one under investigation. To measure the effectiveness of the approaches, we decided to use the average estimator. Since the goal of aggregation is to compute at each node an accurate estimate of a global property, for each experiment, we averaged over all the iterations the distance between the highest estimated average by all the nodes and the actual average (i.e. the worst case). Defined the average estimated by the node p at the iteration i as avg_p^i, and the actual average as $ravg^i$, we compute the error as:

$$E = \frac{1}{I} \sum_{i=0}^{I} \frac{max(avg^i) - ravg^i}{ravg^i}$$

where I is the total number of iterations. To be as complete as possible we also evaluated the effects of choosing a different error estimator (i.e. the average case). Experimental data suggested that the qualitative trends don't change. Because of this, all the results presented in the following have been obtained using E.

5.2 Experimental Results

Figures 6, 7, 10 summarize GEA, GEAopt and EVP performances respectively.

The three *(a)* graphs ($n = 1000, \alpha = 0$) show that, for all the three approaches and regardless t, the average error tends to decrease increasing ω because of the number of diffusion-interactions increases as well. More interactions produce an higher convergence speed. Obviously, also the number of messages being exchanged, and than the communication costs, increases with ω.

The three *(b)* graphs ($\omega = 1, n = 1000$) show that – for all the three approaches – the average error decreases quickly by increasing the percent of random links. This outlines the relevance of the network topology in this kind of diffusion-schemes. The more the network is random, the more the network diameter decreases, the lower the average errors.

The three *(c)* graphs ($\omega = 1, \alpha = 0$) show the scalability of the algorithms varying the network size from 100 to 10000 nodes. As expected, for all the three approaches, the average error increases with the number of participating nodes. This happens because the number of iterations needed by the schemes to converge increases with the number of nodes. However, a sublinear trend guarantees good scalability. For the sake of clarity, in all the experiments with epoch-based approaches we used $\gamma = 50$, which is a proper value for the proposed experiments.

Overall, these graphs shows that the three approaches exhibits very similar behavior while changing their common operational parameters, when executing on different network structure, and under different dynamics. The analysis becomes interesting when comparing the relative performances of the three approaches.

5.3 Comparative Evaluation and Summary

To compare the data reported in the previous graphs, in Figure 11 we reported them in a different view. In particular we reported the average error rate of each of the three approaches ($GEA, GEAopt$, EVP) varying the neighborhood density (ω), the number of nodes of the network (n) and the rewiring percent of the network (α). For the sake of clarity we decided to report only the data referring to $t = T(medium)$ because also the other sampling periods produce similar trends.

It is clear from the graphs that GEA is the worst out of the three. Using the same number of messages it produces and average error consistently higher that the other two. EVP is also slightly better that GEAopt. In fact, it produces on average an error 5% lower than GEAopt at no additional communication costs. It is also worth mentioning the effect of the size of the network. With small networks, which have low convergence time, GEAopt outperforms EVP. By

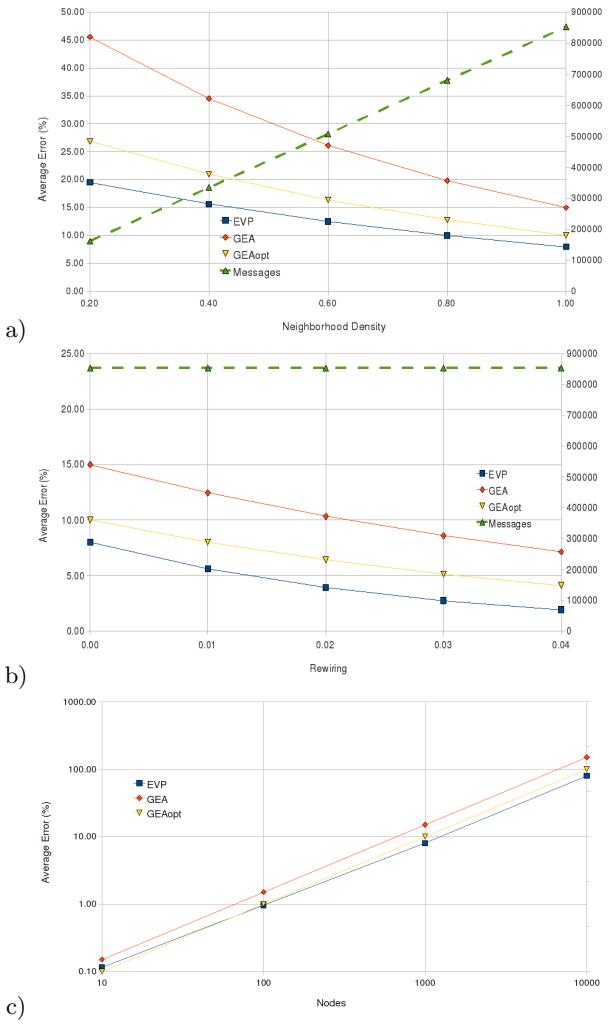

Figure 11: The average error rate of the three approaches ($GEA, GEAopt, EVP$) varying network (a) density ω ($n = 1000, \alpha = 0$), (b) rewiring α ($\omega = 1, \alpha = 0$), (c) size n ($\omega = 1, n = 1000$).

increasing the size the network the result changes, and EVP is able to achieve an error consistently lower. In other words, it appears that EVP is more scalable the GEA and GEAopt.

Our proposal is able to manage the dynamism of the values measured over the network and, in the most of the cases, is more accurate that the other two. In our opinion, it has three key advantages: *(i)* it has no need of periodic restarts and could be taken into account whenever very large networks are involved; *(ii)* it can produce average errors slightly lower that epoch-based solutions which are, moreover, less predictable and inevitably characterized by periodic error spikes as shown in Figure 4; *(iii)* it has interesting autonomic properties, since it does not requires a fine-tuning of its parameters, as depicted in Figure 9.

6. CONCLUSIONS AND FUTURE WORKS

We presented an innovative bio-inspired approach to handle dynamics in diffusive aggregation schemes, relying on evaporation of aggregated values, and have it compared with more traditional epoch-based approaches based on periodic restarts. Our proposal is able to effectively manage dynamics of the values measured over the network and, in the most of the cases, is more accurate that epoch-based approaches. In particular, experimental results show that the proposed evaporative solution is to be preferred in very large networks, in networks with high-sampling rates, and in those particular setups where periodic restarts should be avoided. Yet, for small-size networks, epoch-based solutions work pretty well too.

Regarding open issues, we outline that our current implementation of the evaporative approach is not able to autonomously adapt he evaporation rate to the specific characteristics of the network on which the algorithm is executing. Although this does not undermine the effectiveness of our approach, self-adaptation of the evaporation rate could make it even more effective. In addition, we also think it would be necessary to to investigate how the evaporation mechanism can behave in the presence of different network topologies and in much larger networks than we have tested so far.

7. REFERENCES

[1] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[2] O. Babaoglu, G. Canright, A. Deutsch, G. D. Caro, F. Ducatelle, L. Gambardella, N. Ganguly, M. Jelasity, and R. Montemanni. Design patterns from biology for distributed computing. *In Proceedings of the European Conference on Complex Systems*, 1, 2006.

[3] N. Bicocchi, M. Mamei, and F. Zambonelli. Self-organizing spatial regions for sensor network infrastructures. In *AINA Workshops (2)*, pages 66–71, 2007.

[4] M. Brunato, R. Battiti, and A. Montresor. Gosh! gossiping optimization search heuristics. In *In Proceedings of the Learning and Intelligent Optimization Workshop (LION 2007*, 2007.

[5] P. Costa, V. Gramoli, M. Jelasity, G. P. Jesi, E. L. Merrer, A. Montresor, and L. Querzoni. Exploring the interdisciplinary connections of gossip-based systems. *Operating Systems Review - Special topic: Gossip-Based Networking (OSR)*, 41(4):51–60, oct 2007.

[6] J. Crowcroft. Toward a network architecture that does everything. *Communication of the ACM*, 51(1):74–77, 2008.

[7] D. Estrin, D. Culler, K. Pister, and G. Sukjatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59 – 69, 2002.

[8] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulieacute;. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004.

[9] B. Hayes. Cloud computing. *Communication of the ACM*, 51(7):9–11, 2008.

[10] J. Hong, S. Lu, D. Chen, and J. Cao. Towards bio-inspired self-organization in sensor networks: Applying the ant colony algorithm. *Advanced Information Networking and Applications, International Conference on*, 0:1054–1061, 2008.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, August 2005.

[12] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. Case studies for self-organization in computer science. *Journal of Systems Architecture*, 52(8-9):443–460, 2006.

[13] A. Montresor. Intelligent gossip. In *2nd International Symposium on Intelligent Distributed Computing (IDC 2008)*, Catania, Italy, Sept. 2008. Invited paper.

[14] S. Nath and P. Gibbons. Synopsis diffusion for robust aggregation in sensor networks, 2003.

[15] D. E. Thomas Schoellhammer, Ben Greenstein. Hyper: A routing protocol to support mobile users of sensor networks. Technical report, January 2006.

[16] S. Voulgaris, M. van Steen, and K. Iwanicki. Proactive gossip-based management of semantic overlay networks. *Concurrency and Computation: Practice and Experience*, 19(17):2299–2311, December 2007.