A Self-Organized clustering scheme for overlay networks

Francois Cantin, Bamba Gueye, Mohamed Ali Kaafar, Guy Leduc

University of Liege, Belgium
{francois.cantin, cabgueye, ma.kaafar, guy.leduc}@ulg.ac.be

Abstract. Hierarchical approaches, where nodes are clustered based on their network distances, have been shown to allow for robust and scalable topology-aware overlays. Moreover, recent research works have shown that cluster-based deployments of Internet Coordinates Systems (*ICS*), where nodes estimate both intra-cluster and inter-cluster distances, do mitigate the impact of Triangle Inequality Violations (*TIVs*) on the distance predictions, and hence offer more accurate internet latency estimations. To allow the construction of such useful clusters we propose a self-organized distributed clustering scheme. For better scalability and efficiency, our algorithm uses the coordinates of a subset of nodes, known by running an ICS system, as first approximations of node positions. We designed and evaluated two variants of this algorithm. The first one, based on some cooperation among nodes, aims at reducing the expected time to construct clusters. The second variant, where nodes are selfish, aims at reducing the induced communication overhead.
Keywords: ICS, Clustering, Triangle Inequality Violations, Performance.

1 Introduction

Recent years have seen the advent of Internet applications which are built upon and benefit from topology-aware overlays. In particular, most if not all of these applications and associated overlays rely on the notion of network proximity, usually defined in terms of network delays or round-trip times (RTTs), for optimal neighbor selection. Recent research has focused on proposing elegant solutions to the problem of proximity retrieval while avoiding algorithms that can prove to be very onerous in terms of measurement overheads, and significant bandwidth consumption (ping storms). In this context, network positioning systems, such as [1,2], were introduced. The key idea is that if each node can be associated with a "virtual" coordinate in an appropriate space, distance between nodes can be trivially computed without the overhead of a direct measurement. Despite the elegance from a theoretical perspective, these systems have some practical limitations. In particular, Internet latencies that do violate triangle inequalities, in the actual Internet [3], degrade both coordinates' accuracy and stability [4,5].

Recent research works, however, have shown that shorter internet paths are less likely victims of severe TIVs. Following these observations, in [5] we evaluated the efficiency of a hierarchical approach for ICS. In this approach, nodes that are near each other are clustered, and an independent ICS runs in each cluster. These independent ICS are used to estimate intra-cluster distances, whereas inter-cluster distances are

simply estimated by using an ICS involving all nodes. Since only short paths remain inside the clusters, there are less TIVs in these subspaces. Consequently, the hierarchical ICS offers more accurate latency estimations for intra-cluster distances. More generally, hierarchical overlay approaches, where nodes are clustered based on their network distances, have been shown to allow for robust and scalable network-aware overlays [6,7,8]. In such case, scalability is achieved by drastically reducing the bandwidth requirements and management overhead for overlay maintenance. Moreover, robustness is obtained by mitigating the effect of dynamic environment as most changes are quickly recovered and not seen beyond the clustered set of nodes.

In this paper we propose a self-organized clustering scheme whose goal is twofold. Firstly we address the problem of constructing efficient clusters in an autonomous way by building on an existing ICS system. Secondly our clustering scheme aims at providing a self-managed clustering structure to overlay-based applications, to allow both topology awareness and scalability of these applications. The novelty of our approach lies in simultaneously relying on the partial knowledge of coordinates of nodes involved in ICS operations, and on a distributed clustering algorithm based on adaptive back-off strategy, to construct efficient network topology-aware clusters in a load-balancing way. The main idea is to allow each node to identify a set of clusters in the network, using its own knowledge of a set of nodes' coordinates (as provided by the ICS in which it is involved), and to verify the validity of such clusters using a few measurements towards the identified cluster heads. The distributed algorithm is scheduled using an exponential back-off strategy, where nodes plan their own wake-up time to verify the existence of clusters in their proximity or not. Our main objective behind this strategy is to load balance the clustering process, while adapting to previous clusters creation, and hence optimize the maintenance and measurements overhead.

We provide two variants of our distributed algorithm: a first variant, called "Cooperative", aims at reducing the expected time to construct clusters for the whole network. This approach induces some overhead to inform other nodes that they are likely to belong to a newly created cluster. A second "Selfish variant" is also introduced, where nodes are more selfish and can only form and/or join clusters when they wake up, without any assistance (or guidance) from other nodes that woke up earlier. In both cases nodes use only knowledge provided by a subset of other nodes, in some neighborhood as explained later, and obtain the needed pieces of information (coordinates, existing cluster heads) by piggybacking them in the messages exchanged by the ICS system.

We analyze the performance of our distributed self-clustering algorithm considering clusters efficiency in terms of actual latency existing between members of such clusters, and considering their size. We also observe the time needed to construct efficient clusters, and the induced overhead. By measuring both exchanged messages and measurement rates, we show that our distributed clustering algorithm is fit for purpose, providing a simple, practical and efficient way to build useful topology-aware clusters.

The remainder of the paper is as follows: in section 2, we describe briefly the quality threshold clustering algorithm on which we based our self-organized clustering scheme, and we discuss the reasons that motivate the choice of such an algorithm. In section 3, we introduce the proposed algorithm to allow nodes to identify clusters they belong to. We also discuss the variants of our distributed algorithm, with their pros and cons.

Section 4 presents the clustering results in terms of achieved performance, induced overhead and convergence time. Section 5 concludes the paper.

2 QT (Quality Threshold) clustering algorithm

Clustering is defined as a process of partitioning a set of elements into a number of groups based on a measure of similarity between the data (distance-based approaches) or relying on the assumption that the data come from a known distribution (model-based approaches). For our self-clustering process, we aim at exploiting nodes' coordinates as a first approximation of the inter-node distances existing in the actual network topology. As nodes' coordinates do not follow any a priori distribution, we will focus on distance-based clustering. Moreover, since we aim at providing a self-clustering process that is performed in a distributed way among all the nodes of the network, the optimal number of clusters that can be created is not known in advance. Approaches that do set a constraint on the number of clusters to be formed (such as K-means, C-Means Fuzzy Clustering, etc.) are thus inappropriate.

Having in mind these facts, we choose to leverage the Quality Threshold algorithm (*QT_clustering*) to propose our self-organized clustering scheme. This algorithm has been initially proposed by Heyer et al. [9] for genetic sequence clustering. It is based on the unique constraint of the cluster diameter, as a user-defined parameter. For the QT_clustering and for the remainder of this paper we define the cluster diameter as the maximal distance existing among any two members of the cluster. The QT_clustering is an iterative algorithm and starts with a global set that includes all the elements (*e.g* node coordinates) of the data set, and then returns a set of clusters that respect the quality threshold. Such threshold is defined in terms of the cluster diameter.

First, for each element, a candidate cluster seeded by this element is formed. Such cluster is iteratively added by other elements. Each iteration adds the element that minimizes the increase in cluster diameter. The process continues until no element can be added without surpassing the diameter threshold. A second candidate cluster is formed by starting with the second element and repeating the procedure. Note that all elements are made available to the second candidate cluster. That is, the elements from the first candidate cluster are not removed from consideration. The process continues for all elements. At the conclusion of this stage, we have a set of candidate clusters. The number of candidate clusters is equal to the number of elements, and many candidate clusters overlap. At this point, the largest candidate cluster is selected and retained. The elements it contains are removed from consideration and the entire procedure is repeated on the smaller set. A possible termination criterion is when the largest remaining cluster has fewer elements than some specified threshold.

3 Self-Clustering process

In this section we describe how we exploit the QT_clustering algorithm to provide a distributed self-organized clustering process, based on the knowledge of a subset of nodes' coordinates in a metric space, resulting from running a positioning system to estimate network distances. We will denote by (direct) neighbors the set of peer nodes

that are used as neighbors in the ICS for the purpose of coordinate computation. We will also denote by *long-sight* neighbors, the union of these (direct) neighbors and the neighbors' neighbors (i.e., node's 2-hop neighbors). For instance, if a node has 32 neighbors in order to estimate its coordinates, its long-sight neighbors will be formed by at most 1024 nodes.

3.1 Description

The general idea of our clustering algorithm is to distribute the clustering tasks among nodes in the network relying not only on measurements towards a potential existing cluster, but also on their knowledge of the coordinates of their long-sight neighbors. In other words, if a node wakes up (with respect to the clusters algorithm) and does not find directly an existing cluster it may belong to, it tries to construct such cluster based on the coordinates as provided by the ICS it is running. In such a way, nodes that do wake up earlier try to create clusters that their peers waking up later may join. Put simply, nodes perform trailblazing of the network conditions, to construct the clusters in a distributed way, while optimizing the needed overhead. Three main advantages could then be considered. Firstly nodes do not need global knowledge of nodes in the network, nor distances between these nodes, nor a common landmark/anchor infrastructure. Secondly the network is not overloaded by measurements performed to obtain the cluster structure. And thirdly the network is able to self-construct the clusters that may exist.

During the cluster forming phase, nodes are initially in a waiting mode. Each node waits for an initiator timer according to an exponential random distribution, computed as described in section 3.2. The clustering process follows the procedure presented in Algorithm 1 and can be described as follows: each time a node wakes up, it gets the list of existing cluster heads in the network. Although such information could be obtained by requesting the set of long-sight neighbors that the node is aware of, we choose to perform this information retrieval by exploiting the communication already established at the level of the ICS. Existing cluster heads are propagated in the network by simply piggybacking in the classical ICS messages the identity of the cluster head(s) of cluster(s) a node belongs to. Considering these already existing clusters, each node verifies its membership to one of them. If the measurement towards the cluster head satisfies the cluster diameter, say D, meaning that such distance is less than $D/2$, the node simply joins such cluster by sending a $JOIN$ message to the cluster head. Our previous study [5] showed that when the cluster diameter does not exceed a fixed threshold (*e.g* 140ms), intra-cluster paths are less likely victims of severe TIVs. Following this observation, for our simulations, we set the upper bound of the cluster diameter D to 140ms. Finding a way to adapt automatically this upper bound to the network is one of our future work. Depending on the maximum number of clusters a node can join, say k, such procedure could be repeated with other cluster heads.

Nevertheless if none of the distances to existing cluster heads satisfies the clustering criterion, the node starts the QT-clustering algorithm on the basis of the coordinates of its long-sight neighbors. It is worth noticing that this clustering is just a first approximation. Indeed coordinates may be subject to distance estimations errors, resulting from inaccuracies in coordinates. However this gives the node an approximate view of its

Algorithm 1 Procedure when a node wakes up

```
1: if The node is already in at least one cluster then
2:     The node goes back to sleep;
3: else
4:     The node gets the list of existing cluster heads (known by its long-sight neighbors);
5:     The node measures RTTs to all existing cluster heads;
6:     Let C be the list of existing cluster heads within a range RTT < D;
7:     if C ≠ ∅ then
8:         The node joins at most the k nearest clusters whose heads are in C;
9:     else
10:        Let S be the list of coordinates of the node's long-sight (1-hop and 2-hop) neighbors;
11:        The node runs a QT-Clustering on S ⇒ This returns a set of clusters;
12:        if The node is in none of these clusters then
13:            The node goes back to sleep;
14:        else
15:            The node selects a cluster head in its cluster;
16:            The node measures the RTT to this new potential cluster head;
17:            if RTT > D/2 then
18:                The node goes back to sleep;
19:            else
20:                The node freezes all of its long-sight neighbors (by sending them a message);
21:                if A neighbor answers that it is already frozen by another node then
22:                    The node goes back to sleep;
23:                else
24:                    After a short while the node notifies the selected cluster head and waits for
                       confirmation;
25:                    if Confirmation is positive then
26:                        The node joins the cluster;
27:                    else
28:                        The node goes back to sleep;
```

neighbors positions, and in particular of the clusters that could be formed from this approximation. This first coordinate-based clustering phase allows the node to identify a set of clusters in the metric space of the ICS. This set of clusters is then subject to a verification according to direct measurements.

When a node has verified that its distance to an identified cluster head satisfies the clustering criterion, it decides to inform this potential cluster head that it should create a cluster, and waits for a confirmation. The cluster creation is conditioned by the acceptance of the requested cluster head. In fact, a potential cluster head could refuse to lead a cluster because of load constraints, or more specifically because its actual distance to an already existing cluster head has been considered too short. To this end, when a node is informed that it is a potential cluster head, it measures its distance to the list of cluster heads it is aware of. If at least one of these distances is less than $\alpha \times D/2$, for some $1 < \alpha < 2$, distance between the two cluster heads is considered too short to construct a new cluster, and the request is refused. In this case, the node that identifies this cluster head is informed of this refusal and goes back to sleep. Otherwise, i.e. if the

cluster is created, nodes that wake up later follow this decision and consider the cluster head among the list of existing cluster heads.

The algorithm relies on self-organization of nodes. When a node decides to join a cluster, two variants could drive the process of nodes joining the identified clusters. The first variant, with the main goal of speeding the clustering process, is to inform all identified nodes in a cluster of their potential membership to such cluster, and let them check this fact with direct measurements. The second variant trades off the speed of cluster creation against a reduced measurement overhead. In this case nodes never inform others that they may belong to a newly created cluster, and let them discover this fact when they wake up.

Finally it is worth noticing that the wake-up procedure allows also some adaptation to changes in the network. Since distances in the network may evolve over time, including the distances of nodes towards their identified cluster head(s), a node should not stick to any cluster, and should also verify its membership to additional clusters due to new network conditions. Waking up from time to time, following the distributed scheduling as presented in 3.2, allows them to check their membership to existing clusters, and thereby adapt to changes in network conditions.

3.2 Distributed Scheduling of wake-up timers

During the cluster forming phase, nodes are initially in a waiting mode. Each node waits for an initiator timer according to an exponential random distribution, i.e. $f(t_i) = \lambda_i.e^{-\lambda_i.t_i}$, where $\lambda_i = \lambda_0.(n_i/N_i)$; n_i being the number of non already clustered nearby neighbors, and N_i being the total number of known long-sight nearby neighbors. By nearby nodes we refer to nodes whose coordinates indicate that they are (likely to be) within some specified range. To set the timer according to an exponential random distribution, we set p_t = random(0,1), compute λ_i as described above and let $t_i = (-1/\lambda_i).ln(1 - p_t)$. The wake-up timer could then be computed as $timer = min(t_i, MAX\_Timer)$. From the expression of t_i it is obvious that the timer decreases when λ_i increases. Therefore such timer will ensure that the nodes with more residual non-clustered neighbors have more opportunities to (re)initiate the clustering algorithm, since their timer is more likely to elapse before other nodes. The main idea behind this exponential backoff scheduling is to load-balance the clustering process as initiated by nodes in the network, while optimizing the time needed to construct and join the clusters.

We can also expect that in one MAX_Timer period enough nodes initiate the clustering algorithm. To this end the selection of λ should satisfy the following inequation:

$$Prob(t > MAX\_Timer) \leq 1 - p$$

where p is the expected percentage of nodes initiating the algorithm. Therefore, in such a case

$$\int_{MAX\_Timer}^{+\infty} \mathrm{f}(t)dt \leq 1 - p => \lambda \geq -(ln(1-p)/MAX\_Timer)) \qquad (1)$$

Based on (1) we can calculate λ needed to ensure that a percentage of nodes initiate the algorithm, at least in the initial state, when nodes are not clustered yet. From (1) we can conclude that $\lambda_{min} = -(ln(1-p)/MAX\_Timer)$ is sufficient to ensure this.

4 Analyzing the Clusters

In this section we present the results of an extensive simulation study of the self-organised clustering process. We performed a set of simulations using two datasets: the *p2psim* data, a set of 1740 nodes [10] and *Meridian* data, comprising 2500 nodes [11] to model Internet latencies based on real world measurements.

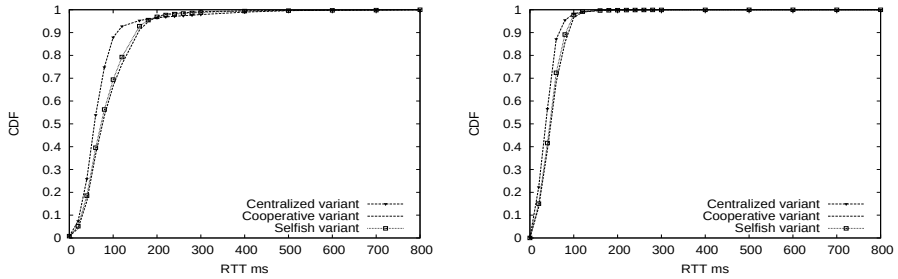
In our simulations, we allowed nodes to join at most two clusters ($k = 2$) and we set the expected maximum cluster diameter D to 140ms for the King dataset and to 80ms for the Meridian dataset, following recommendations in [5]. The maximal timer for a sleeping node is set to 5 minutes and the minimum distance between any two cluster heads is fixed to $3/2 \times D/2$ (*i.e.* $\alpha = 3/2$). As we used coordinates as provided by an ICS in the first step of our self-organised clustering, we deployed the Vivaldi system [2] as a prominent representative of purely P2P coordinate systems. Each node runs the Vivaldi system, setting the number of its neighbors to 32 and our results are obtained for a 2-dimensional coordinate space. In [2], authors show that the more dimensions an Euclidean space has, the more accurate the coordinates are. Similarly, the more neighbors a node has, the more accurate the system is[1]. However, we choose not to illustrate our results by using more accurate coordinates, for ease of deployment and low computing loads, at the cost of some loss in clusters accuracy.

We evaluate the performance of our clustering algorithm with respect to three main indicators. (i) The clusters quality: it is the deviation between the expected cluster diameter and the actual diameter. (ii) The convergence time: it is the time needed by our distributed algorithm to cluster 95% of the nodes in the system. This allows us to differentiate between the initial phase of the algorithm, when clusters are yet in the construction process, and the steady state, when nodes continue to manage their membership to already constructed clusters. Finally, we measured (iii) The overhead: it is the number of exchanged messages and the number of measurements performed. We can further split the overhead during the initial phase and during the steady state. We compare the two variants of our algorithm, and when needed we compare our distributed self-clustering algorithm to a centralized approach. All algorithms used are implemented and evaluated in *R* environment [12].

4.1 Clusters Quality

We can evaluate the cluster quality according to the deviation between the expected cluster diameter, as we set it in the QT_clustering and the actual diameter as obtained after our self-organised clustering process reaches its steady state. However, the cluster size is also an important parameter we should mention. A cluster populated with only a few nodes, even though its diameter is optimal, may be of little use.

[1] As shown in [5], the triangle inequality violations phenomena prevents a perfect mapping between latency and coordinates, even for high-dimensional spaces. Coordinates are deemed to be inaccurate.



(a) King dataset (b) Meridian dataset

Fig. 1. CDF of the RTT of the intra-cluster paths.

To evaluate the clustering quality in terms of cluster diameter, we observe in figures 1(a) and 1(b), the Cumulative Distribution of the actual delays (RTTs) between the members of the same identified cluster (called intra-cluster RTTs). These figures show, for both data sets, the proportion of nodes that actually violate the diameter constraint. We compare the proportion of these violations for the two variants of our clustering algorithm, and for a centralized approach. In this case, a centralized approach consists in emulating a centralized entity that collects the coordinates of all nodes in the system, computes in a centralized way clusters resulting from these coordinates using the QT_clustering and then informs all nodes of the identified cluster heads. These nodes verify their membership to these clusters, and join clusters if the diameter constraint with their cluster heads is verified. Otherwise, they are considered as outliers. The main reason why we compare our algorithm to such a centralized algorithm is to evaluate how partial knowledge of neighborhood and coordinates impact our clustering performance.

Figure 1(a) shows that more than 85% of the intra-cluster links satisfy the cluster diameter constraint, with an RTT less than the expected diameter. The same trend is observed in Figure 1(b) for the Meridian dataset, with more than 95% of clustered nodes scattered in delimited clusters, respecting the expected cluster diameter of 140ms. We also note that both variants are achieving the same performance, which is actually not surprising, since the main difference between our two variants is when nodes join a cluster, and not how they join it. The centralized approach creates slightly more accurate clusters. However, this little difference is overwhelmed by onerous cost induced by a centralized approach that needs global knowledge of both coordinates and nodes in the system.

Performing a QT_clustering based on coordinates of long-sight neighbors gives us a first approximation of nodes positioning. Even though nodes measure network distances, as RTTs, towards identified cluster heads, this does not prevent some mutual distances between cluster members to be above the expected diameter, due to TIVs. Using coordinates reduces the proportion of diameter violations, but since coordinates "only" provide distance estimates, with intrinsic errors, errors may always exist.

As shown in Table 1, the number of clusters identified by our algorithm ranges from 9 to 11 for both variants. However, we can in both cases consider 3 main clusters, with an average population of 700 nodes each for the King dataset, and 1260 nodes as an

average population of each cluster in the Meridian dataset. The percentage of nodes that have not been clustered are roughly 3.8% of nodes existing in the system. The bottom part of Table 1 will be presented later.

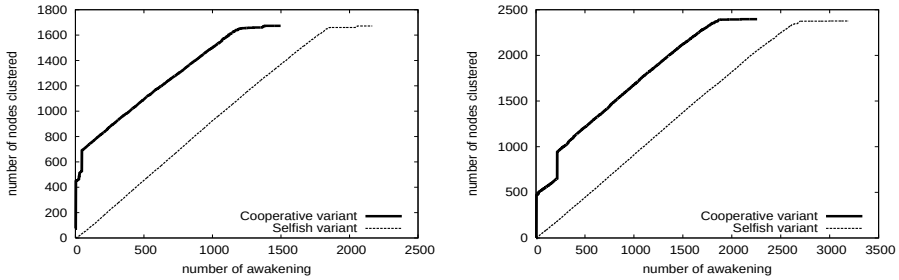
	Cooperative Variant		Selfish Variant	
	King	Meridian	King	Meridian
Number of clusters	9	9	11	9
Number of outliers (unclustered nodes)	67	81	68	102
Total Number of pings	11116	17003	20125	18075
Total Number of messages (excluding pings)	1582	2300	843	246
Convergence time (seconds)	1875	1658	1658	2300
Ping rate before convergence (pings/s)	4.48	8.05	9.95	6.45
Mean ping rate before convergence (pings/node×s)	0.0026	0.003	0.0057	0.0026
Max ping rate before convergence (pings/node×s)	0.027	0.038	0.0398	0.023
Mean msg rate before convergence (msg/node×s)	0.0005	0.0006	0.0003	$4\,10^{-5}$
Max msg rate before convergence (msg/node×s)	0.403	0.635	0.23	0.049
Mean ping rate after convergence (pings/node×s)	0.0002	0.0002	0.0002	0.0002
Max ping rate after convergence (pings/node×s)	0.03	0.032	0.034	0.022
Mean msg rate after convergence (msg/node×s)	10^{-6}	$6\,10^{-7}$	$3\,10^{-7}$	$6\,10^{-7}$
Max msg rate after convergence (msg/node×s)	0.0007	0.0003	0.0002	0.0005

Table 1. Characteristics of the clustering process

4.2 Convergence time

To separate the initial phase from steady state, we analyze the evolution of the number of clustered nodes versus the number of awakenings (and hence versus the number of clustering process calls) for both variants. As depicted in Figures 2(a) and 2(b), the curves labeled "Selfish Variant" follow linear evolutions. Such observation is expected since at most one node can join a cluster at each awakening, and then the growth of the number of clustered nodes can be at best linear. Clusters in the "Cooperative" variant may cumulate node membership with each node's awakening, because information of potential membership to a newly created cluster is sent by the creator of a cluster to identified members. In the curves labeled "Cooperative approach", we can then observe for both data sets the steps corresponding to a set of nodes joining simultaneously a defined cluster. Such steps allow this variant to cluster more than 95% of nodes in 1210 awakenings for the King dataset and in 1854 awakenings for the Meridian dataset, whereas the "Selfish" takes more occurrences, up to 2749 awakenings. Such faster clustering comes at the expense of costs of spreading information and exchanged messages, we will discuss in the following section.

Let us consider that our system is in the steady state if at least 95% of existing nodes have been clustered. Although such parameter relies on our a priori knowledge of the number of outliers in the system, and hence on the minimum number of nodes that can be clustered, it gives us however a suitable way to separate the initial phase from the steady state. It is important to notice that the convergence time, although different in terms of number of awakenings, is roughly the same in real time (in seconds) spent to cluster 95% of the nodes, as shown in table 1. This confirms our choice of the

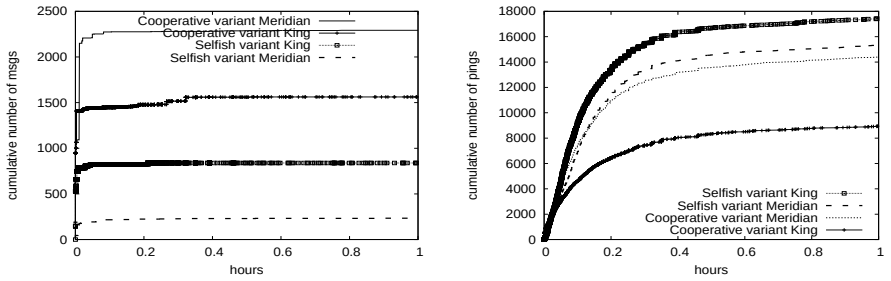


(a) King dataset (b) Meridian dataset

Fig. 2. Evolution of the number of clustered nodes

exponential-backoff strategy to set timers. Recall from section 3.2, that our algorithm will ensure that the greater the λ, the lower the timer, giving hence more opportunity to (re)initiate the clustering algorithm. This guarantees a higher probability to initiate the clustering algorithm in a "area" populated by nodes that have not been clustered yet. This gives a way to adjust the awakening rate according to the number of clustered nodes independently from the number of nodes existing in the system. Such trend is emphasized in the "Cooperative variant" where the convergence time is less than 2000 seconds for both data sets. Next we discuss the cost of the self-clustering algorithm.

4.3 Overhead



(a) Cumulative number of exchanged messages (b) Cumulative number of performed measurements

Fig. 3. Induced Overhead: exchanged messages and performed measurements

To observe the control messages and measurement overhead, we differentiate between the two states of the system: the initial phase (when clusters are built) and the steady state. We observe the induced overhead as the number of measurements performed, but also as the number of exchanged messages during the clustering process. Figure 3(a) depicts the cumulative number of exchanged messages versus time (up to

one hour). We do not consider the $JOIN$ messages sent by clustered nodes to their cluster heads, but focus on the difference that may exist between the two variants of the algorithm.

The sharp rise of the curves in the initial phase is due to the fact that, at the beginning of the algorithm, nodes have the same probability to wake-up, since all nearby neighbors are not clustered yet. We manage to resolve such potential conflictual situation using the Freeze messages, sent to the long-sight neighbors, when a node identifies a cluster head, and waits for its confirmation. We are more likely to encounter such situations at the beginning of the algorithm. Moreover as very few nodes are clustered in the initial phase, more clusters are created, leading to more exchanged messages and measurements.

We can observe from figure 3(a) that, as expected, the cumulative number of messages by the "Selfish variant" is less important than the "Cooperative variant". It is however important to observe that the number of exchanged messages induced by newly created clusters is very low after the initial phase. Once the system reaches the steady state, no more messages, are exchanged. The low message exchange rate observed during the initial phase is confirmed by our results in table 1 with very low message rates of the scale of 10^{-4} per node per second as average values, and a maximum of 0.635 message/node$\times$second.

In figure 3(b) we observe the cumulative number of measurements performed towards the identified cluster heads (typically ping messages). We see again that the major overhead is induced during the initial phase. We observe more pings initially, when nodes initiate their clustering process, with a maximum ping rate of 0.034 ping/node$\times$ second. Such very low overall ping rate per node is promising for large scale deployment of our clustering scheme.

5 Conclusion

We have presented a distributed self-organized clustering process suitable for both ICS accuracy enhancement, and scalable network-aware overlay deployment. We have shown that, although ICS systems suffer from inaccuracies resulting from triangle inequality violations in the Internet, they can be exploited to construct efficient clustering schemes in a distributed way. Indeed, the proposed clustering process is based on a first approximation of node positioning using coordinates, along with an adaptive back-off strategy that allows load-balanced construction of clusters. We also present two variants of such clustering process that trade off the convergence time against the induced overhead. However, the two variants have been shown to be effective, enjoying good clustering performance, while achieving a very good trade-off between scalability and convergence time. Indeed, nodes are able to identify and join their clusters in a reasonable amount of time, while rarely violating the diameter constraints, and they still do not trigger too frequent measurements.

Although this paper focused on Vivaldi for distance estimates and experimentations, the algorithm proposed for clustering is independent of the coordinate system used. Our proposed scheme would then be general enough to be applied in the context of coordinates computed by any Internet coordinate systems.

The reader should note though, that we do not yet consider churn situations, when nodes join and leave the system running the Internet Coordinate System in an asynchronous way. If we consider highly-dynamic networks, the differentiation between the initial phase and the steady state may fade away. However, our clustering process would adapt to situations when only few nodes are existing in the system, by simply not creating clusters if they are "useless". Basically, by setting a minimum number of nodes per cluster at the level of the QT_clustering, low populated clusters could be avoided. It is also important to note that in churn situations, ping and message rates would be moderate, and that considering a non-dynamic network in our simulations gives us a worst-case of the cost of the system, at least in its initial phase. Finally, even though this paper does not address the problem of clusters maintenance, we note that different solutions to such issues have been proposed elsewhere (e.g. [13]). Our future work would then consist in integrating such maintenance techniques in our distributed clustering scheme.

References

1. T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM*, New York, NY, USA, June 2002.
2. F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. ACM SIGCOMM*, Portland, OR, USA, Aug. 2004.
3. H. Zheng, E. K. Lua, M. Pias, and T. Griffin, "Internet Routing Policies and Round-Trip-Times," in *Proc. the PAM Conference*, Boston, MA, USA, Apr. 2005.
4. E. K. Lua and T. Griffin, "Embeddable overlay networks," in *IEEE Symposium on Computers and Communications*, Aveiro, Portugal, 2007.
5. M. A. Kaafar, B. Gueye, F. Cantin, G. Leduc, and L. Mathy, "Towards a two-tier internet coordinate system to mitigate the impact of triangle inequality violations," in *Proc. IFIP Networking Conference*, Singapore, May 2008, LNCS 4982, pp. 397–408.
6. E. K. Lua, X. Zhou, J. Crowcroft, and P. V. Mieghem, "Scalable multicasting with network-aware geometric overlay," *Comput. Commun.*, vol. 31, no. 3, pp. 464–488, 2008.
7. Y. He, Q. Zhao, J. Zhang, and G. Wu, "Topology-aware multi-cluster architecture based on efficient index techniques," in *NPC*, 2005, LNCS 3779, pp. 163–171.
8. G. Xue, Y. Jiang, Y. You, and M. Li, "A topology-aware hierarchical structured overlay network based on locality sensitive hashing scheme," in *UPGRADE*, New York, NY, USA, 2007, pp. 3–8, ACM.
9. L. J. Heyer, S. Kruglyak, and S. Yooseph, "Exploring expression data: Identification and analysis of coexpressed genes," *Genome Research*, vol. 9, no. 11, pp. 1106–1115, nov 1999.
10. *A simulator for peer-to-peer protocols*, `http://www.pdos.lcs.mit.edu/p2psim/index.html`.
11. B. Wong, A. Slivkins, and E. Sirer, "Meridian: A lightweight network location service without virtual coordinates," in *Proc. the ACM SIGCOMM*, aug 2005.
12. R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0.
13. X. Liu, J. Lan, P. Shenoy, and K. Ramaritham, "Consistency maintenance in dynamic peer-to-peer overlay networks," *Comput. Netw.*, vol. 50, no. 6, pp. 859–876, 2006.