A fully distributed clustering algorithm based on random walks

Alain BUI
PR*i*SM (UMR CNRS 8144)
UVSQ, Versailles, France
alain.bui@prism.uvsq.fr

Abdurusul KUDIRETI
CReSTIC-SysCom
URCA, Reims, France
kudireti.abdurusul@univ-reims.fr

Devan SOHIER
PR*i*SM (UMR CNRS 8144), UVSQ, Versailles
and CReSTIC-SysCom – URCA, Reims, France
devan.sohier@univ-reims.fr

Abstract

In this paper, we present a fully distributed clustering algorithm based on random walks that works on arbitrary topologies. A cluster is composed of a set of nodes called the core *that coordinates the clustering process, and of non-core nodes called ordinary nodes. A core is built through a random walk based procedure. Its neighboring nodes that do not belong to any cluster are recruited by the core as ordinary nodes into its cluster.*

*The correctness and termination of our algorithm are proven. We also prove that when two clusters are adjacent, at least one of them has a complete core (*i.e. *a core with the maximum size allowed by the user). Our algorithm is not deterministic, which allows a better load balancing, since the core nodes are not determined by their* ids *and/or location.*

Key words : Clustering, distributed algorithms, random walks.

1. Introduction

Computer networks tend to grow larger and larger. To manage them, one can consider dividing them into several disjoint connected parts: then each part has to be managed separately and these different parts have to be coordinated.

In this work, we focus on the clustering of large networks, which is particularly useful for applications that require scalability to thousands of nodes. A cluster is a connected subset of nodes, and clustering is the process of computing disjoint covering clusters, in our case, in a distributed manner. This clustering should be initiated independently by several nodes in order to speed up the process. It should also make use only of local information on the network, so that it can be fully distributed. All the computed clusters should have roughly the same size.

We propose an original algorithm which is based both on random walks to build the *core* of clusters and a 1-hop *"extension"* of this core: our algorithm provides some guarantees on the size of the core of the clusters. Namely, given a constant $MaxCoreSize$, each cluster core has its size in $[2; MaxCoreSize]$, and no two adjacent clusters can both have their core size smaller than $MaxCoreSize$ (when two clusters are adjacent, at least one has reached its maximum core size).

The main original feature of our algorithm is its use of random walks. The clustering is driven by random walks, thus leading to a random clustering. We make use of the nodes identities only to solve conflicts, rather than basing the clustering on them or on weights (which is the case in many clustering algorithms designed for *ad-hoc* networks).

The paper is organized as follows. The section 2 deals with the state of art on clustering, in particular on *ad hoc* solutions, since many recent works on clustering have focused on this context. Our algorithm is described in detail in section 3. Properties are presented in section 4.

2. Related work

Several distributed algorithms have been proposed during the last years. In this section, we selected the main results related to our study.

D. J. Baker and A. Ephremides presented the algorithm LCA [2]. According this algorithm, a node i becomes a clusterhead if it has the highest *id* among all nodes within 1 hop of it, or when a neighboring node j is such that i is the highest *id* node in j's neighborhood. LCA heuristics were developed to be used with small networks of less than 100 nodes.

Basagni proposed two distributed and mobility-adaptative algorithms: DMAC [3] and GDMAC [4]. In these algorithms, the choice of clusterheads is based on a generic weight instead of node identities or node degrees. More recently, fault-tolerant versions of these algorithms were presented in [7].

The clustering algorithms we have presented so far construct single-hop clusters, *i.e.* clusters so that each node in the network is the neighbor of a clusterhead. In more general terms, multi-hop (k-hop) clustering algorithms (*e.g.* [8], [6]) generate clusters such that any node in any cluster is at most k hops away from the clusterhead.

k-hop cluster can be built through a hierarchical method. At each level i between 1 and k, i-hop clusters are built, in which the clusterheads are the ordinary nodes of the next

978-0-7695-3680-4/09 $25.00 © 2009 IEEE
DOI 10.1109/ISPDC.2009.21

level, and at level $i+1$, the links connected the clusterheads of adjacent clusters.

The solution of Max-Min D-hop-cluster [1] generalizes cluster formation to k-hop clusters. The rules of the Max-Min heuristic are similar to those of LCA, but the former converges on a clusterhead solution much faster at the network layer, with $2k$ rounds of message exchange.

3. Contribution

We propose to base our clustering algorithm on random walks. Indeed, random walks can adapt to arbitrary topologies, and behave well in dynamic networks; the use of random walks helps designing random clusters, which avoid the use of weights and helps balancing the load of the nodes (no node is doomed to be in the core due to its weight and/or location); the use of random walks also avoids using messages broadcasting.

This mechanism builds what we call the core of a cluster. The cluster is composed of its core and of some nodes that are adjacent to it. Thus, it is not necessary to cover all the network only with random walks: we need to cover only a subset of nodes (*core*) in the network by using a *Token* message, and recruit the remaining nodes in this network with the *Recruit* messages, such that every node in this cluster is adjacent to a node of its core. This mechanism is intended to speed up the clustering.

The size of the core is always between 2 and a constant $MaxCoreSize$, which is defined at the beginning of the algorithm. At the end of the clustering, if two clusters are adjacent, at least one of them has a complete core (*ie.* the size is $MaxCoreSize$), and we get these results without using a hierarchical method. This property guarantees that the size of some cluster cores reaches the maximum size.

3.1. Model

A random walk based distributed algorithm is an algorithm involving a particular message, the *token*, that circulates according to a random walk scheme : at each step, one node possesses the token (*ie.* it is the last one that has received the token); after a treatment that only the node holding the token is allowed to proceed to, the node chooses one of its neighbors randomly, and sends the token to it. Random walk-based distributed algorithms do not require any assumption about the topology of the system, and they can often be designed to be tolerant to topological changes.

According to the properties of random walks, each node will eventually be visited by the token. The *covering* and *hitting* properties are also useful for our future work. The average times to achieve these different properties can be computed accurately [5].

3.2. The algorithm

Our algorithm uses *Token* messages that browse the network according to a random walk scheme. Each *Token* message is used to build the core of a cluster: when a node receives the *Token* message, if it does not belong to a cluster core, it chooses to join the cluster core whose *id* is carried by the *Token* message; otherwise, it sends the *Token* message back to its sender. This *Token* message contains the *id* of the cluster, and a counter of nodes which are already recruited, so that the number should not exceed $MaxCoreSize$.

A timeout mechanism ensures that all the nodes, which are not recruited, eventually create their own cluster. However, to limit the number of clusters, it is possible to delete the clusters : when a node in an incomplete core receives a *Token* message from another cluster which has a larger *id* than its cluster *id*, it deletes the core to which it belongs, and joins the core whose *id* is carried by the token it received.

When a node receives the token, it sends a *Recruit* message to all its neighbors. Nodes that do not belong to a cluster respond by choosing to join the cluster of the sender node (they join the cluster and not its core, since the core of one cluster is built only by the *Token* message).

1) *On receiving* Token *message (*cf. *algorithm 1 section "Token")*: When node i receives a *Token*$[P_t, K_t, O_t]$ message (P_t is a cluster of *id* t; K_t is the counter of the core size; O_t is the token order number) from one of its neighbors e: if (P_t, O_t) is in the $to-delete$ list, the node i executes no procedure; otherwise, the node i executes one of the following procedures:
   - Delete the former core (*cf.* R1): If node i is in another core and the *id* of this core is less than the *id* of Core t, and $K_i < MaxCoreSize$, node i joins the core of cluster t and send a *Delete* message to its neighbors, K_i and K_t are updated. If $K_i < MaxCoreSize$, node i will send the *Token*$[P_t, K_t, O_t]$ message to one of its neighbors chosen at random. It will also send the *Recruit* message to all nodes in N_i.
   - Transmit the token (*cf.* R2): If i is already in the core of P_t, and $K_t < MaxCoreSize$, it will transmit the token to one of its neighbors at random.
   - Send the token back to the sender (*cf.* R3): If node i is in the core of another cluster, and the *id* of this core is larger than t, the node i will send the token to e. When $K_t = 1$, i sends a *Delete* message to e.
   - Join the core (*cf.* R4): If the node i does not belong to any cluster core, node i will join the core of cluster t, and $K_i = K_t + 1$, if $K_i < maxCoreSize$, node i will send the *Token* message to one of the neighbors chosen at random and broadcast *Recruit* messages.

2) *On receiving a* Recruit *message (*cf. *algorithm 1 section "Recruit")*: On receiving a *Recruit* message from node j, node i checks if it belongs to any cluster or not. If not, it joins the cluster of j, else node i ignores the message.
3) *On receiving a* Delete *message (*cf. *algorithm 1 section "Delete")*: When i receives a *Delete* message from j, if $K_i = K_t$ and $O_i = O_t$, it leaves the cluster, and broadcasts a *Delete* message; else if node i is a core node, it sends a *Recruit* message to j. The order number guarantees that a newer cluster with the same *id* will not be deleted in the process of deleting the older cluster.

3.3. Illustration of algorithm

The following example presents our clustering algorithm. We take a graph with 20 nodes with unique *ids* (*cf.* figure 1). We suppose that some of the nodes start to send *Token* messages and begin to build the clusters. In this example, the maximal core size is 4 ($MaxCoreSize = 4$).

We suppose that nodes 13, 15, 17, 7 timers expire first. Each of them sends a *Token* message to a neighbor chosen at random.

Node 13 sends $Token[P_{13}, 1, 1]$ to node 3, node 20 sends *Recruit* messages to all its neighbors of N_{13} (same procedure for Node 15,17 and 7, *cf.* second figure of figure 1).

When node 11 receives $Token[P_{15}, 3, 1]$ from node 16, it checks that $K_{11} < MaxCoreSize$, compares the *id* of token ($P_{15} > P_{17}$) and sends the token back to node 16, and continues to building its cluster with $P = 17$ (*cf.* third figure of figure 1). Finally we obtain 4 clusters and two of them are complete (*cf.* last figure of figure 1).

4. Properties

To prove the correctness of our algorithm, we proved the following 5 properties:

1) *The clusters will eventually stabilize.* If they did not stabilize, and since there is only a finite number of possible configuration (by configuration, we mean here the set of clusters), the execution of the algorithm would go twice through the same configuration. Since each cluster modification increases its size, or destroys it, all modified clusters have to be destroyed, and they can only be destroyed by a cluster with a larger *id*. The modified cluster with the largest *id* has thus to be destroyed by a cluster with a larger *id*, which has itself to be modified. This is impossible: the clusters stabilize.
2) *Every node eventually belongs to a cluster.* If i is not in the cluster, when its timer expires, i generates a *Token* message and creates its own cluster.

Algorithm 1 Clusterization algorithm on a node i

Initialization
$P_i \longleftarrow ND$;
$O_i \longleftarrow 0$;
$isCore \longleftarrow false$;
$K = 0$;
$T_i \longleftarrow$ random number in $[n, 2n]$;
$to-delete = \emptyset$
On timeout
$P_i \longleftarrow i$;
$O_i \longleftarrow O_i + 1$;
$isCore \longleftarrow true$;
Send $Token(i, 1, 1)$ to $j \in N_i$ at random;
Send *Recruit* to N_i;
On reception of $Token(P_t, K_t, O_t)$ **from node** e
if $(P_t, O_t) \notin to-delete$ **then**
  if $K_i < MaxCoreSize$ **then**
    if $isCore$ **then**
      if $P_i < P_t$ **then**
        send $Delete(P_i, O_i)$ to N_i; /* ***R1*** */
        $to-delete \longleftarrow to-delete \cup (P_i, O_i)$;
        if $(P_i = i)$ **then**
          $O_t \longleftarrow O_t + 1$;
        end if
        $P_i \longleftarrow P_t$;
        $K_t \longleftarrow K_t + 1$;
        $K_i \longleftarrow K_t$;
        Send $Token(P_t, K_t, O_t)$ to $j \in N_i$ at random;
        Send Recruit to N_i
      else if $(P_i = P_t)$ **then**
        Send $Token(P_t, K_t, O_t)$ to $j \in N_i$ at random; /* ***R2*** */
        $iscore \longleftarrow$ true;
      else
        if $K_t = 1$ **then**
          Send $Delete(P_t, O_t)$ to e; /* ***R3*** */
          Send Recruit to e;
        else
          Send $Token(P_t, K_t, O_t)$ to e;
        end if
      end if
    else
      $P_i \longleftarrow P_t$; /* ***R4*** */
      $K_t \longleftarrow K_t + 1$;
      $K_i \longleftarrow K_t$
      $isCore \longleftarrow true$;
      if $K_t < MaxCoreSize$ **then**
        Send $Token(P_t, K_t, O_t)$ to $j \in N_i$ at random;
      end if
      Inactivate Timeout;
      Send Recruit to N_i;
    end if
  else
    Send $Token(P_t, K_t, O_t)$ to e;
  end if
end if
On reception of *Recruit* message from node j
if $(P_i = ND)$ **then**
  $P_i \longleftarrow P_j$;
end if
On reception of *Delete* message from node j
if $(P_i = P_j)$ **then**
  if $j = P_j$ **then**
    $O_j \longleftarrow O_j + 1$;
  end if
  $to-delete \longleftarrow to-delete \cup (P_j, O_j)$;
  Send $Delete(P_i, O_i)$ to N_i;
  $P_i \longleftarrow ND$;
  $isCore \longleftarrow false$;
  Reset Timeout;
else
  if $isCore$ **then**
    Send *Recruit* to j
  end if
end if

3) *The K_t of any cluster in $[2, MaxCoreSize]$.* $K_t \leq MaxCoreSize$ since a node can be recruited in a core

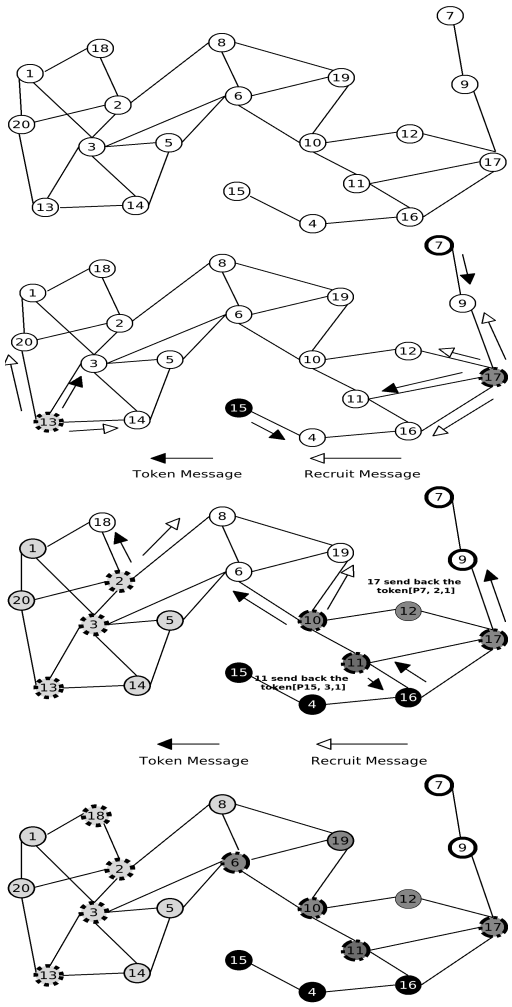

Figure 1. Example

only by a token message, and the K_t variable ensures that no more than $MaxCoreSize$ sites are recruited. $K_t \geq 2$ because if i has a neighbor that is not in a core, i will recruit this neighbor; if all its neighbors are in a core, node i will eventually delete its token and join another core (*cf.* algorithm 3). Thus $K_t \geq 2$.

4) *Two adjacent clusters can not both be incomplete in the steady state.* Suppose two adjacent clusters P_i and P_j are such that $K_i < MaxCoreSize$ and $K_j < MaxCoreSize$ (once the clusters have stabilized). Suppose that $P_i > P_j$. According to the hitting property of random walks, the *Token* message $Token[P_i, K_i, O_i]$ will eventually reach a node in the core of P_j, which will be deleted (it is incomplete according to our assumption), which contradicts the stabilization hypothesis: P_i and P_j can not both be incomplete.

5) *The cluster is connected in the steady state.* By construction, the core is connected (nodes are recruited along the path of a random walk). Ordinary nodes are all recruited by core nodes, and are adjacent to them. Thus the cluster is connected.

5. Conclusion and perspectives

In this paper, we have presented an original algorithm to compute clusters in a network in a distributed way. This algorithm is based on random walks, and requires no assumption on the network topology. This algorithm makes use of random walks to build the *cores* of clusters. The *id*s are used only to break symmetries. Nodes neighboring a core can be *"adopted"*, which speeds up the clustering.

We have proven that no two adjacent clusters can have incomplete cores. This result gives a lower bound on the size of the majority of clusters that have more than one neighbor.

We are now aiming at making this algorithm adaptative to topological changes. The main problem that remains is to manage the loss of connectivity of clusters. Some simulations are currently being run to assert more accurately the expected size of the clusters.

References

[1] T. H. P. V. Alan D. Amis, Ravi Prakash and D. T. Huynh. Max-min d-cluster formation in wireless ad hoc networks. *IEEE INFOCOM*, pages 32–41, 2000.

[2] D. J. Baker and A. Ephremides. Architectural organization of a mobile radio network via a distributed algorithm. *IEEE Transactions on Communications*, 29(11):1694–1701, 1981.

[3] S. Basagni. Distributed clustering for ad hoc networks. *In Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks*, pages 310–315, 1999.

[4] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. *In Proceedings of the 50th IEEE International Vehicular Technology Conference*, pages 889–893, 1999

[5] A. Bui and D. Sohier. How to compute times of random walks based distributed algorithms. *Fundamenta Informaticae, IOS Press*, 80(4): 363–378, 2007.

[6] Y. Fernandes and D. Malkhi. K-clustering in wireless ad hoc networks. *In the 2nd ACM international workshop on Principles of mobile computing*, pages 31–37, 2002.

[7] C. Johnen and L. Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theor. Comput. Sci.*, Elsevier, 410(6-7):581-594, 2009

[8] F. G. Nocetti, J. S. Gonzalez, and I. Stojmenovic. Connectivity based k-hop clustering *In wireless networks. Telecommunication Systems*, 22(1-4) :205–220, 2003.