Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A self-stabilizing k-clustering algorithm for weighted graphs[☆]

Eddy Caron[a,b], Ajoy K. Datta[c], Benjamin Depardon[a,b,*], Lawrence L. Larmore[c]

[a] University of Lyon, LIP Laboratory, UMR CNRS - ENS Lyon - INRIA - UCB Lyon 5668, France
[b] LIP - Équipe GRAAL, 46 allée d'Italie, 69364 Lyon Cedex 07, France
[c] School of Computer Science, University of Nevada, Las Vegas, USA

ARTICLE INFO

Article history:
Received 17 December 2009
Received in revised form
21 June 2010
Accepted 28 June 2010
Available online 24 July 2010

Keywords:
k-clustering
Self-stabilization
Weighted graph

ABSTRACT

Mobile *ad hoc* networks as well as grid platforms are distributed, changing, and error prone environments. Communication costs within such infrastructure can be improved, or at least bounded, by using *k-clustering*. A k-clustering of a graph, is a partition of the nodes into disjoint sets, called clusters, in which every node is distance at most k from a designated node in its cluster, called the *clusterhead*. A self-stabilizing asynchronous distributed algorithm is given for constructing a k-clustering of a connected network of processes with unique IDs and weighted edges. The algorithm is comparison based, takes $O(nk)$ time, and uses $O(\log n + \log k)$ space per process, where n is the size of the network. To the best of our knowledge, this is the first solution to the k-clustering problem on weighted graphs.

1. Introduction

Overlay structures of distributed systems require taking into account locality among the entities they manage. For example, communication time between resources is the main performance metric in many systems. A cluster structure facilitates the spatial *reuse of resources* to increase system capacity. Clustering also helps routing and can improve the efficiency of a parallel software if it runs on a cluster of well connected resources. Another advantage of clustering is that many changes in the network can be made locally, *i.e.*, restricted to particular clusters.

Many applications require that entities are grouped into clusters according to a certain distance function which measures proximity with respect to some relevant criterion; the clustering will result in clusters with similar readings. We are interested in two particular fields of research which can make use of resource clustering: mobile *ad hoc* networks (MANET) and application deployment on grid environments.

In MANET, scalability of large networks is a critical issue. Clustering can be used to design a low-hop backbone network in MANET with routing facilities provided by clustering. However, using hops, *i.e.*, the number of links in the path between two processes, as the sole measure of distance may hide the true communication time between two nodes.

A major aspect of grid computing is the deployment of grid middleware. Hop distance is used as a metric in some applications, but it may not be relevant in some platforms, such as grids. Using an arbitrary metric (*i.e.*, a weighted metric) is a reasonable option in such heterogeneous distributed systems. Distributed grid middleware, like Diet [4] and GridSolve [17] can make use of accurate distance measurements to do efficient job scheduling.

Another important consideration is that both MANET and grid environments are highly dynamic systems: nodes can join and leave the platform anytime, and may be subject to errors. Thus, designing an efficient fault-tolerant algorithm which partitions nodes into clusters which lie within a given distance of each other, and which can dynamically adapt to any change, is valuable for many applications, including MANET and grid platforms.

Self-stabilization [8] is a desirable property of fault-tolerant systems. A self-stabilizing system, regardless of the initial states of the processes and initial messages in the links, is guaranteed to converge to the intended behavior in finite time. As MANET and grid platforms are dynamic and error prone infrastructures, self-stabilization is a very desirable property for the algorithms which manage those structures.

In this paper, we address the problem of clustering a dynamic and error prone network into clusters within which no node is further apart than a given distance k to a specific node of the cluster.

[☆] This work was developed with financial support from the ANR (Agence Nationale de la Recherche) through the SPADES project referenced 08-ANR-SEGI-025.

[*] Corresponding author at: LIP - Équipe GRAAL, 46 allée d'Italie, 69364 Lyon Cedex 07, France.

E-mail addresses: eddy.caron@ens-lyon.fr (E. Caron), datta@cs.unlv.edu (A.K. Datta), benjamin.depardon@ens-lyon.fr (B. Depardon), larmore@cs.unlv.edu (L.L. Larmore).

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.06.009

1.1. The k-clustering problem

We now formally define the problem solved in this paper. Let $G = (V, E)$ a connected graph (network) consisting of n nodes (processes), with positively weighted edges. For any $x, y \in V$, let $w(x, y)$ be the *distance* from x to y, defined to be the least weight of any path from x to y. We will assume that the edge weights are positive integers. The radius of a graph G is defined as follows:

$$radius(G) = \min_{x\in V} \max_{y\in V}\{w(x, y)\}.$$

Given a positive integer k, we define a *k-cluster* of G to be a non-empty connected subgraph of G of radius at most k: such that all processes in the cluster are within distance k of a designated leader process, called the *clusterhead*.

We define a *k-clustering* of G to be a partitioning of V into k-clusters. The *k-clustering problem* is then the problem of finding a k-clustering of a given graph.[1] In this paper, we require that a k-clustering specifies one node, which we call the *clusterhead* within each cluster, which is within k of all nodes of the cluster, and a *shortest path tree* rooted at the clusterhead which spans all the nodes of the cluster.

A set of nodes $D \subseteq V$ is a *k-dominating set*[2] of G if, for every $x \in V$, there exists $y \in D$ such that $w(x, y) \le k$. A k-dominating set determines a k-clustering in a simple way; for each $x \in V$, let $Clusterhead(x) \in D$ be the member of D that is closest to x. Ties can be broken by any method, such as by using IDs. For each $y \in D$, $C_y = \{x : Clusterhead(x) = y\}$ is a k-cluster, and $\left\{C_y\right\}_{y\in D}$ is a k-clustering of G.

We say that a k-dominating set D is *optimal* if no k-dominating set of G has fewer elements than D. The problem of finding an optimal k-dominating set, or equivalently, a k-clustering with the minimum possible number of clusters, is known to be $\mathcal{NP}$-hard [1]. Our algorithm attempts to find a k-clustering which has "few" clusters.

1.2. Related work

Amis et al. [1] give the first distributed solution to this problem. The time and space complexities of their solution are $O(k)$ and $O(k \log n)$, respectively. Spohn and Garcia-Luna-Aceves [16] give a distributed solution to a more generalized version of the k-clustering problem. In their algorithm, a parameter m is given, and each process must be a member of m different k-clusters. The k-clustering problem discussed in this paper is then the case $m = 1$. The time and space complexities of the distributed algorithm in [16] are not given. Fernandess and Malkhi [10] give an algorithm for the k-clustering problem that uses $O(\log n)$ memory per process, takes $O(n)$ steps, provided a *Breadth First Search* (BFS) tree[3] for the network is already given.

The first self-stabilizing solution to the k-clustering problem was given by Datta et al. in [6]; this solution takes $O(k)$ rounds and $O(k \log n)$ space. Another stabilizing solution was proposed in [5]; this algorithm needs $O(n)$ rounds and $O(\log n)$ space. Both solutions use the hop metric, and are thus unable to deal with more general weighted graphs.

Many algorithms have been proposed in the literature for constructing clusters in distributed network. Other self-stabilizing clustering algorithms deal with weighted graphs where weights are placed on the vertices, not on the edges. For example, Johnen and Nguyen give in [12] an algorithm to partition the network into 1-hop clusters, *i.e.,* the algorithm computes a *dominating set*, a set S such that ever node is a neighbor of some member of S. The article presents self-stabilizing versions of DMAC [2] and GDMAC [3]. The authors also give a robust version of both algorithms in [13], *i.e.,* after one round the network is partitioned into clusters, and stays partitioned during construction of the final clusters.

A self-stabilizing algorithm for cluster formation under a *density* criterion is presented in [14] by Mitton et al. The density criterion (defined in [15]) is used to select clusterheads — a node v is elected a clusterhead if it has the highest density in its neighborhood, and the cluster headed by v contains all nodes at distance less or equal to two from v.

1.3. Contributions

Our solution, Algorithm K-CLUSTERING , given in Section 6, is partially inspired by that of Amis et al. [1], who use hop distance instead of arbitrary edge weights. K-CLUSTERING uses $O(\log n + \log k)$ bits per process. It finds a k-dominating set in a network of processes, assuming that each process has a unique ID, and that each edge has a positive weight. It is also self-stabilizing and converges in $O(nk)$ rounds. Even though this convergence time may seem high, our theoretical analysis and simulations results, presented respectively in Sections 7 and 8, show that this bound is reached on special graphs, and that in practice, less rounds are required to converge. To the best of our knowledge, this is the first solution to the k-clustering problem on weighted graphs.

As our solution is a combination of several self-stabilizing algorithms, we also present the conditions under which the combination of self-stabilizing algorithms is also self-stabilizing under the unfair daemon.

1.4. Outline

In Section 2, we describe the model of computation used in the paper, and give some additional needed definitions. We iteratively build our solution. We first present a non-self-stabilizing algorithm for the "Best Reachable Problem" in Section 3, which is central in our solution. Then, we make this algorithm self-stabilizing in Section 4. Our self-stabilizing algorithm for the k-clustering problem being in fact composed of four self-stabilizing algorithms, we present in Section 5 the conditions for composing self-stabilizing algorithms under the unfair daemon. We are then able to present K-CLUSTERING , a self-stabilizing algorithm for the k-clustering problem on weighted graphs in Section 6, and give its complexity in number of rounds, and the worst case scenario for the number of clusterheads in Section 7. Finally, we present some simulation results in Section 8 before concluding the paper in Section 9.

2. Preliminaries

We consider a connected undirected network of n processes, where $n \ge 2$, and an integer $k \ge 1$. Each process P has a unique ID, $P.id$ of an ordered type, which we call ID type. The *state* of a process is defined by the values of its registers. A *configuration* of the network is a function from processes to states; if γ is the current configuration, then $\gamma(P)$ is the current state of each process P. An *execution* of an algorithm, $\mathcal{A}$ is a sequence of states $e = \gamma_0 \mapsto \gamma_1 \mapsto \cdots \mapsto \gamma_i \cdots$, where $\gamma_i \mapsto \gamma_{i+1}$ means that it is possible for the network to change from configuration γ_i to configuration γ_{i+1} in one step. We say that an execution is *maximal* if it is infinite, or if it ends at a *sink*, *i.e.,* a configuration from which no execution is possible.

[1] There are several alternative definitions of k-clustering, or the k-clustering problem, in the literature.

[2] Note that this definition of the *k-dominating set* is different than another well known problem consisting in finding a subset $V' \subseteq V$ such that $|V'| \le k$, and such that $\forall v \in V - V', \exists y \in V' : (x, y) \in E$ [11].

[3] A BFS tree has a designated *root*, and from each node, the path from that node through the BFS tree to the root is the shortest possible path in the network.