Self-stabilizing Neighborhood Unique Naming under Unfair Scheduler

Maria Gradinariu and Colette Johnen

Laboratoire de Recherche en Informatique, UMR CNRS 8623,
Université de Paris Sud, F91405 Orsay cedex, France
{mariag,colette}@lri.fr

Abstract.
We propose a self-stabilizing probabilistic solution for the neighborhood unique naming problem in uniform, anonymous networks with arbitrary topology. This problem is important in the graph theory Our solution stabilizes under the unfair distributed scheduler. We prove that this solution needs in average only one trial per processor.
We use our algorithm to transform the [6] maximal matching algorithm self-stabilizing to be able to cope up with a distributed scheduler.

1 Introduction

Self-stabilization. Self-stabilization introduced by Dijkstra, [2], provides an uniform approach to fault-tolerance, [9]. More precisely, this technique guarantees that, regardless of the initial state, the system will eventually converge to the intended behavior without the need for explicit exception handler of backward recovery. In this paper we are particular interested in uniform (every processor in the system executes the same algorithm) and anonymous systems (processors does not have a distinct identifier).

Neighborhood Unique Naming (NUN) problem. In [5] is defined the labeling graphs problem with conditions at distance 2. NUN problem, issued from this theoretical graph problem, ensures that in each neighborhood the vertex have distinct labels. In other words, there is no vertex with the same label as one of its neighbors and there is no vertex having neighbors named identically. This problem is a classical coloring problem with an additional restriction: the vertex at distance two have distinct labels. A practical application is the assigning radio frequencies to transmitters such that transmitters that are close (distance 1 or 2 apart) to each other use different frequencies.

Maximal Matching (MAM) problem. The MAM problem is issued from graph theory. A *matching* in a graph is a set of edges where no two edges are adjacent. The matching set M is called maximal if there is no other matching set M' such that $M \subset M'$. The main applications of this problem in distributed computing area are job assignment and task scheduling.

Related works. The classical vertex coloring problem is a restriction of the NUM problem. The vertex coloring was previously studied for planar and bipartite graphs (see [3, 14, 12, 13]). Using a well-known result from graph theory, Gosh and Karaata [3] provide an elegant solution for coloring acyclic planar graphs with exactly six colors, along with an identifier based solution for acyclic orientation of planar graphs. This makes their solution limited to systems whose communication graph is planar and processors have unique identifiers. Sur and Srimani [14] vertex coloring algorithm is only valid for bipartite graphs. A paper by Shukla *et al.* (see [13]) provides a randomized self-stabilizing solution to the two coloring problem for several classes of bipartite graphs, namely complete odd-degree bipartite graphs and tree graphs. In [4] the authors presents three coloring algorithms for the arbitrary networks. Their solutions use $O(D)$ colors, where D is the maximal degree of the network.
Nevertheless, all the previous presented algorithms are not solutions for the NUN problem since it may be possible that vertex at distance 2 are labeled identically. The NUN has multiple applications such as: the acyclic orientation of general networks or finding the maximal matching sets. The first application is trivial, therefore we focus on the MAM problem. There are several works treating the MAM problem. The faster known sequential algorithm is a wave algorithm due to Micali and Vazirani, [7]. This solution is not self-stabilizing.
A deterministic self-stabilizing solution for the MAM problem was provided by Huang and Hsu in [6]. Their solution stabilizes only under a central scheduler.

Our contribution. We present the first self-stabilizing solution for the NUM problem. Our solution works on anonymous, uniform networks with any topology and it needs in average only one trial per processor under the distributed scheduler. (I.E. a processor randomly updates its local identifier one time on the average). This algorithm is used to transform the [6] MAM algorithm such that it stabilizes under a distributed scheduler. Note that in the transformed algorithm the randomization is used only for breaking the symmetry, once any processor of the network gets an unique local identifier, the NUN algorithm has no further influence (no action is performed), the MAM algorithm evolution is then deterministic. Our solution copes up with the most powerful scheduler — the distributed scheduler : only the processors with a locally maximal identifier can choose their match. Hence, we avoid the matching cycles generation (more details in Section 5).

2 Model

Distributed Systems. A distributed system is a set of state machines called processors. Each processor can communicate with some processors called neighbors. A processor p communicates to its neighbors via its variables that its neighbors can read but cannot update. We will use $\mathcal{N}_p$ to denote the set of neighbors of the processor p.

A processor p in a distributed system executes an algorithm which is a finite set of guarded actions where each guard is a boolean expression over its variables and the variables of its neighbors, and where each statement is a deterministic or probabilistic update of the local and variables of p.

The state of a processor p is the value of its variables. A *configuration* of a distributed system is an instance of the processor states. A processor is *enabled* in a given configuration if at least one of the guards of its algorithm is *true*. During a computation step, one or more enabled processors perform the statement of an enabled action. A *computation* of a distributed system DS is a *maximal* sequence of computations steps. *Maximality* means that the sequence is either infinite, or the terminal configuration is a deadlock.

A distributed system can be modeled by a transition system. A transition system is a three-tuple $S = (\mathcal{C}, \mathcal{T}, \mathcal{I})$ where $\mathcal{C}$ is the collection of all configurations, $\mathcal{I}$ is a subset of $\mathcal{C}$ called the set of initial configurations, and $\mathcal{T}$ is a function $\mathcal{T}$ from $\mathcal{C}$ to the set of $\mathcal{C}$ subsets. A $\mathcal{C}$ subset of $\mathcal{T}(c)$ is called a c transition. An element of a c transition t, is called an output of t. In a probabilistic system, there is a probabilistic law on the output of a transition; in a deterministic system, each transition has only one output.

Scheduler. In this model, a *scheduler* is a *predicate* over the system computations. In a computation, a transition (c_i, c_{i+1}) occurs due to the execution of a nonempty subset of the enabled processors in the configuration c_i. In every computation step, this subset is chosen by the scheduler : a *central* (resp. *distributed*) scheduler chooses one and only one enabled processor (resp. a subset of the enabled processors) to execute a statement action. A scheduler may be unfair.

A strategy is the set of computations that can be obtained under a specific scheduler choice. At the initial configuration, the scheduler "chooses" one set of enabled processors (it chooses a transition). For each output of the selected transition, the scheduler chooses a second transition, and so on. The strategy formal definition is based on the tree of computations. Let c be a system configuration. A *TS-tree* rooted in c, $\mathcal{T}ree(c)$, is the tree-representation of all computations beginning in c. Let n be a node in $\mathcal{T}ree(c)$ (i.e. a configuration), a *branch* rooted in n is the set of all $\mathcal{T}ree(c)$ computations starting in n with a computation step of the same n transition. The degree of n is the number of branches rooted in n. A *sub-TS-tree of degree* 1 rooted in c is a restriction of $\mathcal{T}ree(c)$ such that the degree of any $\mathcal{T}ree(c)$'s node (configuration) is at most 1. Let st be a strategy of the distributed system DS, an *st-cone* $\mathcal{C}_h$ is the set of all possible st-computations with the same prefix h (for more details see [10]). The last configuration of h is denoted $last(h)$. We have equipped a strategy with a probabilistic space (see [1] for more details). The measure of an st-cone $\mathcal{C}_h$ is the measure of the prefix h (i.e., the product of the probability of every computation step occurring in h).

Probabilistic self-stabilization. Let DS be a distributed system. A predicate P is closed for the computations of DS if and only if when P holds in a configuration c, P also holds in any configuration reachable from c. Let D be a scheduler and st be a strategy of DS under D. Let CP be the set of all system configurations satisfying a closed predicate P (formally $\forall c \in CP, c \vdash P$). The set of st-computations that reach configurations of CP is denoted by $\mathcal{EP}$ and its probability by $P_{st}(\mathcal{EP})$.

In this paper we study quasi-silent algorithms : those for which a legitimate configuration (i.e. it verifies the problem specification) is also a deadlock. A quasi-silent probabilistic self-stabilizing algorithm is not silent because it may have infinite executions (these executions do not converge). The measure of these executions is null, in any strategy.

Definition 1 (Probabilistic Stabilization of a quasi-silent algorithm). *A distributed system $\mathcal{DS}$ is* self-stabilizing *under a scheduler D for a specification $\mathcal{L}$ (a legitimacy predicate on configurations) such that in any strategy st of $\mathcal{S}$ under D, the following conditions hold :*

- *The probability of the set of st-computations, that reach a configuration satisfying $\mathcal{L}$ is 1. Formally, $\forall st, P_{st}(\mathcal{EL}) = 1$.*
- *A configuration satisfies $\mathcal{L}$ iff it is a deadlock.*

Convergence of Probabilistic Stabilizing Systems. Building on previous works on probabilistic automata (see [11, 15, 10, 8],), [1] presented a framework for proving self-stabilization of probabilistic distributed systems. In the following we recall a key property of the system called *local convergence* and denoted by LC. Let DS be a distributed system. Let st be a strategy of DS, $PR1$ and $PR2$ be two closed predicates on configurations of DS. Let $\mathcal{C}_h$ be a st-cone where $last(h) \vdash PR1$. In st, from $last(h)$, if the probability to reach a configuration that verifies $PR2$ in at most N computation steps is greater or equal than ϵ then we say that $\mathcal{C}_h$ satisfies $LC(PR1, PR2, \epsilon, N)$.
If in strategy st, there exist $\delta_{st} > 0$ and $N_{st} \geq 1$ such that any st-cone, $\mathcal{C}_h$ with $last(h) \vdash PR1$, satisfies $LC(PR1, PR2, \epsilon_{st}, N_{st})$, then the main theorem of [1] states that the probability of the set of st-computations reaching configurations satisfying $PR1 \wedge PR2$ is 1.

3 Impossibility Results

Unique Local Naming (ULN) problem is to ensure that neighbor processors have distinct identifiers but processors at distance 2 may have the same identifier.

Lemma 1. *There is no deterministic self-stabilizing algorithm solving the ULN or the NUM problem in uniform and anonymous networks under distributed scheduler.*

Proof. We will study a ring of n processors running such an algorithm. Let c_0 be a configuration where all processors are in the same state. c_0 is illegitimate : in c_0, at least one processor may execute an action. Thus, all processors are able to execute the same action. After the computation step where all processors have performed the same action, the obtained configuration is symmetrical : all processors are in the same state.

4 Self-stabilizing NUN under a Distributed Scheduler

In the current section, we present a self-stabilizing probabilistic solution for the NUN problem. Algorithm 4.1 idea is very simple. Each processor has a variable, referred in the algorithm as *lid*, which indicates its local identifier and a $flag$ which signals the presence of at least two neighbors having the same *lid*. The space complexity for Algorithm 4.1 is $O(log(n))$ bits per processor. A processor which cannot execute $\mathcal{A}_2$, having at least two neighbors with the same local identifier, l, sets its $flag$ to l (Action $\mathcal{A}_1$). A processor having a neighbor with the same value of *lid* chooses randomly a new identifier from a bounded set of values (Action $\mathcal{A}_2$). The same action is executed by the processor, p, when it has a neighbor, q, which flag value is set to p's *lid*. In this case, the processor q has at least two neighbors (p and another one) with the same *lid*.

Algorithm 4.1 NUN algorithm on the processor i

Constant :
B : integer constant proven optimal for the value $2n^2$
Variable :
$flag.i$: a positive integer bounded by B
Actions :

$$\begin{aligned}\mathcal{A}_1 \ : & (\forall j \in \mathcal{N}.i \ (lid.i \neq lid.j) \wedge (lid.i \neq flag.j)) \wedge \\ & (\exists (j,k) \in \mathcal{N}.i \ (lid.j = lid.k) \wedge (flag.i \neq lid.j)) \wedge \\ & (\nexists \ (t,l) \in \mathcal{N}.i, \ (lid.t = lid.l = flag.i)) \longrightarrow flag.i = lid.j; \\ \mathcal{A}_2 \ : & \exists j \in \mathcal{N}.i \ (lid.i = lid.j) \vee (lid.i = flag.j) \longrightarrow lid.i = random(1, \ldots, B);\end{aligned}$$

Definition 2. *Let p be a processor. $Correct\_lid(p)$ is the following predicate on configurations: (i) $\forall q$ neighbor of p, $lid.p \neq lid.q$; and $lid.p \neq flag.q$; (ii) $\forall r$ neighbor of a p's neighbor, $lid.p \neq lid.r$. Let c be a configuration. We name m_c the number of processors p which does not verify $Correct\_lid(p)$. c is legitimate iff c satisfies the predicate $\mathcal{L}ID \equiv (m_c = 0)$.*

Observation 1 *In Algorithm 4.1, one may prove that a configuration is legitimate iff it is deadlock. In any computation from a configuration c there are at most $n - m_c$ consecutive computation steps in which Action $\mathcal{A}_2$ is not executed.*

Scena is the following scenario described informally by : "at each computation step, where at least one processor performs Action $\mathcal{A}_2$, (i) exactly one of these processors verifies the $Correct\_lid$ predicate after this computation step, (ii) the other processors keep their previous *lid* value". Let c be a configuration. On any computation following the scenario *Scena*, the predicate $Correct\_lid$ is closed : once a processor p verifies $Correct\_lid(p)$, no action of p, of p's neighbor, or of a neighbor of a p's neighbor will change that.

Lemma 2. *Let st be a strategy under a distributed scheduler on a system executing the Algorithm 4.1. There exist $\epsilon > 0$ and $N > 0$ such that all cone of st satisfy LC ($true$, $\mathcal{L}ID$, ϵ, N).*

Proof. Let $\mathcal{C}_h$ be a cone of the strategy *st* such that $last(h) = c$ does not verify $\mathcal{L}ID$. Assume that $m_c \neq 0$. After at most $n - m_c$ computation steps a processor executes Action $\mathcal{A}_2$. We study the scenario *Scena*
Let p be the processor which will change its *lid* value from the configuration c. Let d_p^1 and d_p^2 the numbers of neighbors at distance 1 and 2 of p and let $(lid_j)_{j=1,d_p^1+d_p^2}$ be their local identifiers and let $(flag_k)_{k=1,d_p^1}$ be the flag values for the neighbors at distance 1. The probability that p chooses a value equal to lid_j or $flag_k$ is $\frac{1}{B}$. The probability that the new chosen value be different of all *lid* values of its neighbors at distance 1 or 2 and of $flag$ values of its neighbors at distance 1 is $1 - (\sum_{j=1}^{j=d_p^1+d_p^2} P_{st}(lid_p = lid_j) + \sum_{j=1}^{j=d_p^1} P_{st}(lid_p = flag_j)) = 1 - \frac{2d_p^1+d_p^2}{B} \leq 1 - \frac{2(n-1)}{B}$.
The probability of the computation step that we have defined is greater than $\frac{1}{B}^{(m-1)}(1-\frac{2(n-1)}{B})$ (The probability that a processor keeps its *lid* value, after $\mathcal{A}_2$ is $\frac{1}{B}$). m_c has decreased by 1, hence in at most $m-1$ similar sequences of computation steps, a legitimate configuration is obtained. Thus in *st*, from $\mathcal{C}_h$, the probability to reach a legitimate configuration in at most $N = n^2$ computation steps, is greater than $\epsilon = \left(\frac{1}{B}^{\frac{n}{2}}(1 - \frac{2n}{B})\right)^{n-1}$.

Theorem 1. *Algorithm 4.1 is self-stabilizing under a distributed scheduler for the NUN specification.*

Lemma 3. *When $B > 2n(n-1)$ where n is the system size, a processor performs in the average only one time the randomized Action $\mathcal{A}_2$.*

Proof. Let c be the initial configuration of Algorithm 4.1. The probability that after the action $\mathcal{A}_2$, a processor has an unique local name is $1-\frac{2d_p^1+d_p^2}{B}$. Note that $2d_p^1 + d_p^2$ could be bounded by $2(n-1)$. In the average $m_c(1-\frac{2(n-1)}{B})$ processors have an unique local identifier after all processors that do not verify $Correct\_lid$ have performed one time $\mathcal{A}_2$. $B > 2n(n-1)$ guarantees that in average, all processors have an unique local name after at most one Action $\mathcal{A}_2$.

Algorithm 5.1 MAM algorithm on the processor i

Constants :
B : integer constant proven optimal for the value $2n^2$
Variables :
$lid.i$: a positive and no-null integer bounded by B
$match.i$: a positive integer bounded by B
Actions :
$\mathcal{A}_1$: $(match.i = 0) \wedge (lid.i > max(lid.k, k \in \mathcal{N}.i \wedge match.k = 0) \wedge$
$(\exists j \in \mathcal{N}.i,\ match.j = 0) \wedge (\forall k \in \mathcal{N}.i, match.k \neq lid.i) \longrightarrow match.i = lid.j$
$\mathcal{A}_2$: $(match.i = 0) \wedge (\exists j \in \mathcal{N}.i,\ match.j = lid.i) \longrightarrow match.i = lid.j$
$\mathcal{A}_3$: $(match.i = lid.j) \wedge (match.j \neq 0) \wedge (match.j \neq lid.i) \longrightarrow match.i = 0$
$\mathcal{A}_4$: $(match.i \notin \{lid.j \mid j \in \mathcal{N}.i\} \bigcup \{0\}) \longrightarrow match.i = 0$

5 Self-stabilizing MAM under a Distributed Scheduler

Definition 3 (Matched and Inactive processors). *A processor p is unmatched iff $match.p = 0$. Two neighbors $(p,\ q)$ are matched iff $match.p = q$ and $match.q = p$. A processor p is inactive iff all its neighbors are matched and $match.p = 0$. A legitimate configuration satisfies the predicate $\mathcal{L}ID$ and all processors are inactive or matched.*

Observation 2 *In a legitimate configuration, the states of match variables define a maximal matching. Let c be a deadlock configuration of Algorithm 5.1. In c, if $match.p \neq 0$ then the processor p is matched. Thus, the deadlock configurations are legitimate.*

Theorem 2. *Any computation starting in a configuration satisfying the predicate $\mathcal{L}ID$ is finite.*

Proof (outline). Let e be a computation starting in a configuration satisfying the predicate $\mathcal{L}ID$. Assume that e is an infinite computation. e has an infinite suffix e' where no processor performs the action $\mathcal{A}_4$ or the action $\mathcal{A}_2$. Between two consecutive actions $\mathcal{A}_1$, a processor performs one and only one time the action $\mathcal{A}_3$. Let cs be a computation step along e' where a processor p performs the action $\mathcal{A}_1$ to choose the processor q as "match". Before the computation step, q is unmatched. During the computation step and after that q does not perform any action. Therefore, along e', a processor p performs at most one action $\mathcal{A}_1$ (one can prove that along e' no action $\mathcal{A}_1$ is performed).

6 Conclusion

We present the first algorithm for NUN problem, self-stabilizing on anonymous and uniform systems. Our solution copes up with distributed schedulers and is

time optimal, more precisely we guarantee that the NUN is done in only one trial per processor in the average under any unfair scheduler. The presented solution is used as substratum in the modification of the [6] MAM algorithm such that the new algorithm stabilizes under any distributed scheduler.

References

1. Beauquier, J., Gradinariu, M., and Johnen, C.: Randomized self-stabilizing optimal leader election under arbitrary scheduler on rings. Technical Report 1225, Laboratoire de Recherche en Informatique (1999)
2. Dijkstra, E.: Self stabilizing systems in spite of distributed control. *Communications of the ACM*, vol. 17 (1974) 643-644
3. Ghosh, S., and Karaata, M.H.: A self-stabilizing algorithm for coloring planar graphs. *Distributed Computing*, 7 (1993) 55-59.
4. Gradinariu, M., and Tixeuil, S.: Tight space uniform self-stabilizing l-mutual exclusion. Technical Report 1249, Laboratoire de Recherche en Informatique (2000)
5. Griggs, J. R., and Yeh., R. K.: Labeling graphs with a condition at distance two. *SIAM, Journal of Discrete Mathematics*, 5 (1992) 586-595
6. Hsu, S., and Huang, S.: A self-stabilizing algorithm for maximal matching. In *Information Processing Letters*, 43(2) (1992) 77-81
7. Micali, S., and Vazirani, V.: An algorithm for finding maximum matching in general graphs. In *21st IEEE Annual Symposium on Foundations of Computer Science* (1980)
8. Pogosyants, A., Segala, R., and Lynch N.: Verification of the randomized consensus algorithm of Aspen and Herlihy: a case study. In *Distributed Computing*, 13 (2000), 155-186
9. Schneider M.: Self-stabilization. *ACM Computing Surveys*, 25 (1993), 45-67
10. Segala, R.: *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, Dep. of Electrical EnG. and Comp. Science (1995)
11. Segala, R., and Lynch, N.: Probabilistic simulations for probabilistic processes. In LNCS, *CONCUR '94, Concurrency Theory, 5th International Conference*, Vol. 836 (1994)
12. Shukla, S., Rosenkrantz, D., and Ravi, S.: Developing self-stabilizing coloring algorithms via systematic randomization. In *Proc. of the Int. Workshop on Parallel Processing* (1994) 668-673
13. Shukla, S., Rosenkrantz, D., and Ravi, S.: Observations on self-stabilizing graph algorithms for anonymous networks. In *Proc. of the Second Workshop on Self-stabilizing Systems*, pages 7.1–7.15 (1995)
14. Sur S., and Srimani P. K.: A self-stabilizing algorithm for coloring bipartite graphs. *Information Sciences*, 69 (1993) 219-227
15. Wu, S. H., Smolka, S. A., and Stark, E. W.: Composition and behaviors of probabilistic i/o automata. In LNCS, *CONCUR '94, Concurrency Theory, 5th International Conference*, Vol. 836 (1994) 513-528