Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On probabilistic snap-stabilization ☆

Karine Altisen [a,b], Stéphane Devismes [a,b,*]

[a] Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
[b] CNRS, VERIMAG, F-38000 Grenoble, France

ARTICLE INFO

Article history:
Received 21 March 2014
Received in revised form 27 July 2015
Accepted 2 August 2015
Available online xxxx

Keywords:
Snap-stabilization
Probabilistic algorithms
Leader election

ABSTRACT

In this paper, we introduce *probabilistic snap-stabilization*. We relax the definition of deterministic snap-stabilization without compromising its safety guarantees. In an unsafe environment, a probabilistically snap-stabilizing algorithm satisfies its safety property *immediately* after the last fault; whereas its liveness property is only ensured with probability 1.

We show that probabilistic snap-stabilization is more expressive than its deterministic counterpart. Indeed, we propose two probabilistic snap-stabilizing algorithms for a problem having no deterministic snap- or self-stabilizing solution: *guaranteed service leader election* in arbitrary anonymous networks. This problem consists in computing a correct answer to each process that initiates the question "Am I the leader of the network?," *i.e.*, (1) processes always compute the same answer to that question and (2) exactly one process computes the answer *true*.

Our solutions being probabilistically snap-stabilizing, the answers are only delivered within an almost surely finite time; however any delivered answer is correct, regardless the arbitrary initial configuration and provided the question has been properly started.

1. Introduction

Self-stabilization [2] is a versatile technique to withstand *any* transient fault in a distributed system: a self-stabilizing algorithm is able to recover, *i.e.*, reach a legitimate configuration, in finite time, regardless the *arbitrary* initial configuration of the system, and therefore also after the occurrence of transient faults. Thus, self-stabilization makes no hypotheses on the nature or extent of transient faults that could hit the system, and recovers from the effects of those faults in a unified manner. Such versatility comes at a price. After transient faults, there is a finite period of time, called the *stabilization phase*, before the system returns to a legitimate configuration. During this phase, there is no safety guarantee at all. In addition, a process cannot locally detect whether the system is actually in a legitimate configuration. Moreover, self-stabilizing algorithms may require a large amount of resources, *e.g.*, extra memory is usually required to crosscheck inconsistencies. Finally, symmetries occurring in the initial configuration could cause a problem to be impossible to solve, *e.g.*, leader election [3] and token passing [4] have no deterministic self-stabilizing solutions in anonymous networks. To cope with those issues, two categories of variants of self-stabilization have been introduced: *weakened* and *strengthened* forms of self-stabilization.

☆ A preliminary version of this work has been published in ICDCN'2014 [1].

* Corresponding author.
E-mail addresses: Karine.Altisen@imag.fr (K. Altisen), Stephane.Devismes@imag.fr (S. Devismes).

http://dx.doi.org/10.1016/j.tcs.2015.08.001
0304-3975/

1.1. Related work

Weakened forms of self-stabilization have been introduced to cope with impossibility results, reduce the stabilization time, or limit the resource consumption. *Weak stabilization* [5] stipulates that starting from *any* initial configuration, *there exists* a run that eventually reaches a legitimate configuration. Unlike for self-stabilization, token passing and leader election have weak stabilizing solutions in anonymous networks [6]. *k-Stabilization* [7] prohibits some of the configurations from being initial, as an initial configuration may only be the result of at most k faults. There are k-stabilizing token passing algorithms that guarantee small convergence time depending only on k [7]. *Probabilistic self-stabilization* [8] weakens the convergence property: starting from any initial configuration, the system converges to a legitimate configuration with probability 1. Problems such as token passing and leader election in anonymous networks have probabilistic self-stabilizing solutions [8,9].

Strengthened forms of self-stabilization have been mainly introduced to offer stringent safety guarantees. A *fault containing* self-stabilizing algorithm [10] ensures that when few faults hit the system, the faults are both spatially and temporally contained. "Spatially" means that if only few faults occur, those faults cannot be propagated further than a preset radius around the corrupted nodes. "Temporally" means quick stabilization when few faults occur. A *superstabilizing algorithm* [11] is self-stabilizing and has two additional properties. In presence of single topological change, it recovers fast, and a safety predicate, called a *passage* predicate, should be satisfied along the stabilization. Finally, *(deterministic) snap-stabilization* [12] offers strong safety guarantees: regardless of the configuration to which transient failures drive the system, after the failures stop, a snap-stabilizing system immediately resumes correct behavior. Precisely, a snap-stabilizing algorithm guarantees that any computation started after the faults will operate correctly. However, we have no guarantees for those executed all or a part during faults. Actually, snap-stabilization is often used to offer *user-centric* guarantees: the problems considered consist of executing finite tasks called *services*: a service is started by some initiating process and terminates by providing a result to that initiator. The goal is to ensure that, starting from any configuration, a service eventually starts if requested by some process; and every started service is computed correctly. We call those problems *guaranteed service problems*.

1.2. Contribution

We introduce a new property called *probabilistic snap-stabilization*, a probabilistic variant of (deterministic) snap-stabilization. Just as for the probabilistic extension of self-stabilization, we choose to adopt a "Las Vegas" approach and relax the definition of snap-stabilization without altering its safety guarantees. (We will also investigate an alternative definition following the Monte Carlo approach.) Considering a specification as the conjunction of safety and liveness properties, a probabilistically snap-stabilizing algorithm immediately satisfies the safety property at the end of the faults, whereas the liveness property is ensured with probability 1.

We show that probabilistic snap-stabilization is strictly more expressive than its deterministic counterpart, as we give two probabilistic snap-stabilizing algorithms for a problem having no deterministic self- or snap-stabilizing solution: *guaranteed service leader election* in anonymous networks. This problem consists in computing a correct answer to each process that initiates the question "Am I the leader of the network ?", *i.e.*, (1) processes always compute the same answer to that question and (2) exactly one process computes the answer *true*. Our solutions being probabilistically snap-stabilizing, the answers are delivered within an almost surely finite time; however, any delivered answer is correct, regardless of the arbitrary initial configuration and provided that the question has been properly started.

Our two algorithms work in the locally shared memory model. The first solution, $\mathcal{S}_{\mathcal{GSLE}}$, assumes a synchronous daemon. The second, $\mathcal{A}_{\mathcal{GSLE}}$, assumes an unfair (distributed) daemon, the most general daemon of the model. Both algorithms need an additional assumption[1]: the knowledge of a bound B such that $B < n \leq 2B$, where n is the number of processes. The memory requirement of both algorithms is in $O(\log n)$ bits per process. The expected delay, response, and service times of $\mathcal{S}_{\mathcal{GSLE}}$ are each $O(n)$ rounds, while these times are $O(n^2)$ rounds for $\mathcal{A}_{\mathcal{GSLE}}$. If we add the assumption that processes know an upper bound, D, on the diameter of the network, the expected time complexity of $\mathcal{S}_{\mathcal{GSLE}}$ (resp. $\mathcal{A}_{\mathcal{GSLE}}$) can be made $O(D)$ rounds (resp. $O(D.n)$ rounds).

1.3. Roadmap

In the next section we define the computational model. In Section 3, we introduce *probabilistic snap-stabilization*. In Section 4, we formally define the guaranteed service problems, and give one example, namely guaranteed service leader election. In Section 5, we propose $\mathcal{S}_{\mathcal{GSLE}}$, our probabilistic snap-stabilizing algorithm for synchronous anonymous systems. In Section 6, we propose $\mathcal{A}_{\mathcal{GSLE}}$, our probabilistic snap-stabilizing algorithm for asynchronous anonymous systems. In Section 7, we investigate an alternative definition of the probabilistic snap-stabilization following the Monte Carlo approach. We also discuss how to adapt our algorithm to efficiently achieve this variant. We conclude in Section 8.

[1] We otherwise prove that our problem is unsolvable.

Please cite this article in press as: K. Altisen, S. Devismes, On probabilistic snap-stabilization, Theoret. Comput. Sci. (2015), http://dx.doi.org/10.1016/j.tcs.2015.08.001