J. Parallel Distrib. Comput. 123 (2019) 26–45

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Gradual stabilization☆

Karine Altisen [a], Stéphane Devismes [a,*], Anaïs Durand [a], Franck Petit [b]

[a] VERIMAG UMR 5104, Université Grenoble Alpes, France
[b] LIP6 UMR 7606, INRIA, UPMC Sorbonne Universités, France

HIGHLIGHTS

- We introduce the notion of gradual stabilization.
- We propose a gradually stabilizing unison algorithm.
- We study the necessity of the system assumptions made for our algorithm.

ARTICLE INFO

Article history:
Received 1 July 2017
Received in revised form 15 June 2018
Accepted 3 September 2018
Available online xxxx

MSC:
68W15
68M15

Keywords:
Self-stabilization
Synchronization problems
Unison
Gradual stabilization
Superstabilization
Safe-convergence

ABSTRACT

We consider *dynamic distributed systems*, *i.e.*, distributed systems that can suffer from topological changes over the time. Following the superstabilizing approach, we assume here that topological changes are transient events. In this context, we introduce the notion of *gradual stabilization under* (τ, ρ)*-dynamics* (gradual stabilization, for short). A gradually stabilizing algorithm is a self-stabilizing algorithm with the following additional feature: after up to τ *dynamic steps* of a given type ρ occur starting from a legitimate configuration, it first quickly recovers to a configuration from which a specification offering a minimum quality of service is satisfied. It then gradually converges to specifications offering stronger and stronger safety guarantees until reaching a configuration (1) from which its initial (strong) specification is satisfied again, and (2) where it is ready to achieve gradual convergence again in case of up to τ new dynamic steps of type ρ. A gradually stabilizing algorithm being also self-stabilizing, it still recovers within finite time (yet more slowly) after any other finite number of transient faults, including for example more than τ arbitrary dynamic steps or other failure patterns such as memory corruptions. We illustrate this new property by considering three variants of a synchronization problem respectively called *strong*, *weak*, and *partial* unison. We propose a self-stabilizing unison algorithm which achieves gradual stabilization in the sense that after one dynamic step of a certain type BULCC (such a step may include several topological changes) occurs starting from a configuration which is legitimate for the strong unison, it maintains clocks almost synchronized during the convergence to strong unison: it satisfies partial unison immediately after the dynamic step, then converges in at most one round to weak unison, and finally re-stabilizes to strong unison.

1. Introduction

In 1974, Dijkstra [9] introduced *self-stabilization*, a general paradigm to enable the design of distributed systems tolerating *any* finite number of transient faults.[1] Consider the first configuration after all transient faults cease. This configuration is arbitrary, but no other transient faults will ever occur from this configuration. By abuse of language, this configuration is referred to as *arbitrary initial configuration* of the system in the literature. Then, a self-stabilizing algorithm (provided that faults have not corrupted its code) guarantees that starting from such an arbitrary initial configuration, the system recovers *within finite time*, without any external intervention, to a so-called *legitimate configuration* from which its specification is satisfied. Thus, self-stabilization makes no hypotheses on the nature (*e.g.*, memory corruptions or topological changes) or extent of transient faults that could hit the system, and the system recovers from the effects of those faults in a unified manner. Such versatility comes at a price, *e.g.*, after transient faults cease, there is a finite period of time, called the *stabilization phase*, during which the safety properties of the system are violated. Hence, self-stabilizing algorithms are mainly compared according to their *stabilization time*, the maximum duration of the stabilization phase. For many problems, the stabilization time is significant,

☆ This study has been partially supported by the ANR projects DESCARTES, France (ANR-16-CE40-0023) and ESTATE, France (ANR-16-CE25-0009).

* Corresponding author.
E-mail addresses: Karine.Altisen@univ-grenoble-alpes.fr (K. Altisen), Stephane.Devismes@univ-grenoble-alpes.fr (S. Devismes), Anais.Durand@univ-grenoble-alpes.fr (A. Durand), Franck.Petit@lip6.fr (F. Petit).

[1] Transient faults have low frequency and results in perturbing the state of the system.

https://doi.org/10.1016/j.jpdc.2018.09.002
0743-7315/

e.g., for synchronization problems [2] and more generally for non-static problems [13] (such as token passing or broadcast) the lower bound is $\Omega(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network. By definition, the stabilization time is impacted by worst case scenarios. Now, in most cases, transient faults are sparse and their effect may be superficial. Recent research thus focuses on proposing self-stabilizing algorithms that additionally ensure drastically smaller convergence times in favorable cases.

Defining the number of faults hitting a network using some kind of Hamming distance (the minimal number of processes whose state must be changed in order to recover a legitimate configuration), variants of the self-stabilization paradigm have been defined, *e.g.*, a *time-adaptive* self-stabilizing algorithm [25] additionally guarantees a convergence time in $O(k)$ time-units when the initial configuration is at distance at most k from a legitimate configuration.

The property of *locality* consists in avoiding situations in which a small number of transient faults causes the entire system to be involved in a global convergence activity. Locality is, for example, captured by *fault containing* self-stabilizing algorithms [14], which ensure 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 processes. "Temporally" means quick stabilization when few faults occur.

Some other approaches consist in providing convergence times *tailored by the type of transient faults*. For example, a *superstabilizing algorithm* [11] is self-stabilizing and has two additional properties when transient faults are limited to a single topological change. Indeed, after adding or removing one link or process in the network, a superstabilizing algorithm recovers fast (typically $O(1)$ rounds), and a safety predicate, called a *passage* predicate, should be satisfied all along the stabilization phase.

Contribution. In this paper, we consider distributed systems that can suffer from topological changes over the time, also referred to as *dynamic distributed systems* in [11]. Following the superstabilizing approach, we assume here that topological changes are transient events. In this context, we introduce a specialization of self-stabilization called *gradual stabilization under* (τ, ρ)*-dynamics*. An algorithm is gradually stabilizing under (τ, ρ)-dynamics if it is self-stabilizing and satisfies the following additional feature. After up to τ *dynamic steps*[2] of type ρ[3] occur starting from a legitimate configuration, a gradually stabilizing algorithm first quickly recovers a configuration from which a specification offering a minimum quality of service is satisfied. It then gradually converges to specifications offering stronger and stronger safety guarantees until reaching a configuration (1) from which its initial (strong) specification is satisfied again, and (2) where it is ready to achieve gradual convergence again in case of up to τ new dynamic steps of type ρ. Of course, the gradual stabilization makes sense only if the convergence to every intermediate weaker specification is fast.

We illustrate this new property by considering three variants of a synchronization problem respectively called *strong*, *weak*, and *partial* unison. In these problems, each process should maintain a local clock. We restrict here our study to periodic clocks, *i.e.*, all local clocks are integer variables whose domain is $\{0, \ldots, \alpha - 1\}$, where $\alpha \geq 2$ is called the *period*. Each process should regularly increment its clock modulo α (liveness) while fulfilling some safety requirements. The safety of strong unison imposes that at most two consecutive clock values exist in any configuration of the system. Weak unison only requires that the difference between clocks of every two neighbors is at most one increment. Finally, we define partial unison as a property dedicated to dynamic systems, which only enforces the difference between clocks of neighboring processes present before the dynamic steps to remain at most one increment.

We propose a self-stabilizing strong unison algorithm $\mathcal{SU}$ which works with any period $\alpha \geq 2$ in an anonymous connected network of n processes. $\mathcal{SU}$ assumes the knowledge of two values μ and β, where μ is any value greater than or equal to $\max(2, n)$, α should divide β, and $\beta > \mu^2$. $\mathcal{SU}$ is designed in the locally shared memory model and assume the distributed unfair daemon, the most general daemon of the model. Its stabilization time is at most $n+(\mu+1)\mathcal{D}+1$ rounds, where n (resp. $\mathcal{D}$) is the size (resp. diameter) of the network.

We then slightly modify $\mathcal{SU}$ to make it gradually stabilizing under (1, BULCC)-dynamics. In particular, the parameter μ should now be greater than or equal to $\max(2, N)$, where N is a bound on the number of processes existing in any reachable configuration. Our gradually stabilizing variant of $\mathcal{SU}$ is called $\mathcal{DSU}$. Due to the slight modifications, the stabilization time of $\mathcal{DSU}$ is increased by one round compared to the one of $\mathcal{SU}$. The condition BULCC restricts the gradual convergence obligation to dynamic steps, called BULCC-dynamic steps, that fulfill the following conditions. A BULCC-dynamic step may contain several topological events, *i.e.*, link and/or process additions and/or removals. However, after such a step, the network should (1) contain at most N processes, (2) stay connected, and (3) if $\alpha > 3$, every process which joins the system should be linked to at least one process already in the system before the dynamic step, unless all of those have left the system. Condition (1) is necessary to have finite periodic clocks in $\mathcal{DSU}$. We show the necessity of condition (2) to obtain our results whatever the period is, while we proved that condition (3) is necessary for our purposes when the period α is fixed to a value greater than 5. Finally, we exhibit pathological cases for periods 4 and 5, in case we do not assume condition (3).

$\mathcal{DSU}$ is gradually stabilizing because after one BULCC-dynamic step from a configuration which is legitimate for the strong unison, it immediately satisfies the specification of partial unison, then converges to the specification of weak unison in at most one round, and finally retrieves, after at most $(\mu + 1)\mathcal{D}_1 + 1$ additional rounds (where $\mathcal{D}_1$ is the diameter of the network after the dynamic step), a configuration (1) from which the specification of strong unison is satisfied, and (2) where it is ready to achieve gradual convergence again in case of another dynamic step.

Notice that $\mathcal{DSU}$ being also self-stabilizing (by definition), it still converges to a legitimate configuration of the strong unison after the system suffers from arbitrary other kinds of transient fault including, for example, several arbitrary dynamic steps. However, in such cases, there is no safety guarantees during the stabilization phase.

Related work. Gradual stabilization is related to two other stronger forms of self-stabilization, namely, *safe-converging self-stabilization* [19] and *superstabilization* [11]. The goal of a safely converging self-stabilizing algorithm is to first quickly (within $O(1)$ rounds is the usual rule) converge from an arbitrary configuration to a *feasible* legitimate configuration, where a minimum quality of service is guaranteed. Once such a feasible legitimate configuration is reached, the system continues to converge to an *optimal* legitimate configuration, where more stringent conditions are required. Hence, the aim of safe-converging self-stabilization is also to ensure a gradual convergence, but only for two specifications. However, such a gradual convergence is stronger than ours as it should be ensured after any step of transient faults,[4] while the

[2] *N.b.*, a dynamic step is a step containing topological changes.

[3] Precisely, ρ is a binary predicate over graphs, representing network topologies, such that $\rho(G, G')$ is true if and only if it is possible for the system to switch from topology G to topology G' in a single (dynamic) step.

[4] Such transient faults may include topological changes, but not only.