J. Parallel Distrib. Comput. 87 (2016) 26–42

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

Snap-stabilizing committee coordination

Borzoo Bonakdarpour[a], Stéphane Devismes[b,*], Franck Petit[c]

[a] Department of Computing and Software, McMaster University, Canada
[b] VERIMAG UMR 5104, Université Joseph Fourier, Grenoble, France
[c] LIP6 UMR 7606, UPMC Sorbonne Universités, Paris, France

HIGHLIGHTS

- We give snap-stabilizing algorithms for the committee coordination problem (CC).
- Our algorithms achieve other properties, e.g., fairness and maximal concurrency.
- We show that there is no CC algorithm satisfying fairness and maximal concurrency.
- We provide 2 snap-stabilizing solutions implementing these properties separately.

ARTICLE INFO

Article history:
Received 17 June 2013
Received in revised form
2 September 2015
Accepted 18 September 2015
Available online 30 September 2015

Keywords:
Distributed algorithms
Snap-stabilization
Self-stabilization
Committee coordination

ABSTRACT

In the *committee coordination problem*, a committee consists of a set of professors and committee meetings are synchronized, so that each professor participates in at most one committee meeting at a time. In this paper, we propose two *snap-stabilizing* distributed algorithms for the committee coordination. *Snap-stabilization* is a versatile property which requires a distributed algorithm to efficiently tolerate transient faults. Indeed, after a finite number of such faults, a snap-stabilizing algorithm immediately operates correctly, without any external intervention. We design snap-stabilizing committee coordination algorithms enriched with some desirable properties related to *concurrency*, *(weak) fairness*, and a stronger synchronization mechanism called 2-*Phase Discussion*. In our setting, all processes are identical and each process has a unique identifier. The existing work in the literature has shown that (1) in general, fairness cannot be achieved in committee coordination, and (2) it becomes feasible if each professor waits for meetings infinitely often. Nevertheless, we show that even under this latter assumption, it is impossible to implement a fair solution that allows *maximal concurrency*. Hence, we propose two orthogonal snap-stabilizing algorithms, each satisfying 2-phase discussion, and either maximal concurrency or fairness. The algorithm that implements fairness requires that every professor waits for meetings infinitely often. Moreover, for this algorithm, we introduce and evaluate a new efficiency criterion called the *degree of fair concurrency*. This criterion shows that even if it does not satisfy maximal concurrency, our snap-stabilizing fair algorithm still allows a high level of concurrency.

1. Introduction

Distributed systems are often constructed based on an asynchrony assumption. This assumption is quite realistic, given the principle that distributed systems must be conveniently expandable in terms of size and geographical scale. It is, nonetheless, inevitable that processes running across a distributed system often need to synchronize for various reasons, such as exclusive access to a shared resource, termination, agreement, rendezvous, *etc.* Implementing synchronization in an asynchronous distributed system has always been a challenge, because of obvious complexity and significant cost; if synchronization is handled in a centralized fashion using traditional shared-memory constructs such as barriers, it may turn into a major bottleneck, and, if it is handled in a fully distributed manner, it may introduce significant communication overhead, unfair behavior, and be vulnerable to numerous types of faults.

* Corresponding author.
E-mail addresses: borzoo@mcmaster.ca (B. Bonakdarpour), stephane.devismes@imag.fr (S. Devismes), franck.petit@lip6.fr (F. Petit).
URLs: http://www.cas.mcmaster.ca/~borzoo/ (B. Bonakdarpour), http://www-verimag.imag.fr/~devismes/ (S. Devismes), http://pagesperso-systeme.lip6.fr/Franck.Petit/ (F. Petit).

http://dx.doi.org/10.1016/j.jpdc.2015.09.004
0743-7315/© 2015 Elsevier Inc. All rights reserved.

The classic *committee coordination problem* [10] characterizes a general type of synchronization called n-ary *rendezvous* as follows:

"*Professors in a certain university have organized themselves into committees. Each committee has an unchanging membership roster of one or more professors. From time to time a professor may decide to attend a committee meeting; he starts waiting and remains waiting until a meeting of a committee of which he is a member is started. All meetings terminate in finite time. The restrictions on convening a meeting are as follows:* (1) *meeting of a committee may be started only if all members of that committee are waiting, and* (2) *no two committees can meet simultaneously, if they have a common member. The problem is to ensure that* (3) *if all members of a committee are waiting, then a meeting involving some member of this committee is convened*".

In the context of a distributed system, professors and committees can be mapped onto *processes* and *synchronization events* (*e.g.*, rendezvous) respectively. Moreover, the three properties identified in this definition are known as (1) Synchronization, (2) Exclusion, and (3) Progress, respectively.

Most of the existing algorithms that solve the committee coordination problem [2,3,10,23,24,26] overlook properties that are vital in practice. Examples include satisfying fairness or reaching maximum concurrency among convened committees and/or professors in a meeting. Moreover, to our knowledge, none of the existing algorithms is resilient to the occurrence of faults. These features are significantly important when a committee coordination algorithm is implemented to ensure distributed mutual exclusion in code generation frameworks, such as process algebras, *e.g.*, CSP, Ada, and BIP [6].

With this motivation, in this paper, we propose snap-stabilizing [7,8] distributed algorithms for the committee coordination problem, where all processes are identical and each process has a unique identifier. Snap-stabilization is a versatile property which requires a distributed algorithm to efficiently tolerate transient faults. Indeed, after a finite number of such faults (*e.g.*, memory corruptions, message losses, *etc.*), a snap-stabilizing algorithm immediately operates correctly, without any external (*e.g.*, human) intervention. A snap-stabilizing algorithm is also a *self-stabilizing* [16] algorithm that stabilizes in 0 steps. In other words, our algorithms are optimal in terms of stabilization time, *i.e.*, every meeting convened *after* the last fault satisfies every requirement of the committee coordination. By contrast, an algorithm that would be only self (but not snap) stabilizing only recovers a correct behavior in finite time after the occurrence of the last fault. Nevertheless, to the best of our knowledge, the committee coordination problem was never addressed in the area of self-stabilization. Therefore, the algorithms proposed in this paper are also the first self-stabilizing committee coordination protocols.

Our snap-stabilizing committee coordination algorithms are enriched with other desirable properties. These properties include Professor Fairness, Maximal Concurrency, and 2-Phase Discussion. The former property means that every professor which requests to participate in a committee meeting that he is a member of, eventually does. Roughly speaking, the second of the aforementioned properties consists in allowing as many committees as possible to meet simultaneously. The latter (2-Phase Discussion) requires professors to collaborate for a minimum amount of time before leaving a meeting.

We first consider Maximal Concurrency and Professor Fairness. As in [23], to circumvent the impossibility of satisfying fairness [24], each time we consider professor fairness in the sequel of the paper, we assume that every professor waits for a meeting infinitely often. Under this assumption, we show that Maximal Concurrency and Professor Fairness are two mutually exclusive properties, *i.e.*, it is impossible to design a committee coordination algorithm (even non-stabilizing) that satisfies both features simultaneously.

Consequently, we focus on the aforementioned contradictory properties independently by providing the two snap-stabilizing algorithms. The former maximizes concurrency at the cost of not ensuring professor fairness. On the contrary, the second algorithm maintains professor fairness, but maximal concurrency cannot be guaranteed. Both algorithms are based on the straightforward idea that coordination of the various meetings must be driven by a priority mechanism that helps each professor to know whether or not he can participate in a meeting. Such a mechanism can be implemented using a token circulating among the professors. To ensure fairness, when a professor holds a token, he has the higher priority to convene a meeting. He then retains the token until he joined the meeting. In that case, some neighbors of the token holder can be prevented from participating in other meetings so that the token holder eventually does. This results in decreasing the level of concurrency. In order to guarantee maximal concurrency (but at the risk of being unfair), a waiting professor must release the token if he is not yet able to convene a meeting to give a chance to other committees in which all members are already waiting.

Thus, in the first algorithm, we show the implementability of committee coordination with Maximal Concurrency even if professors are not required to wait for meetings infinitely often. To the best of our knowledge this is the first committee coordination algorithm that implements maximal concurrency. Moreover, the algorithm is snap-stabilizing and satisfies 2-Phase Discussion.

We also propose a snap-stabilizing algorithm that satisfies Fairness on professors (respectively, committees) and respects 2-Phase Discussion. As mentioned earlier, this algorithm assumes that every professor waits for a meeting infinitely often. Following our impossibility result, the algorithm does not satisfy Maximal Concurrency. However, we show that it still allows a high level of concurrency. We analyze this level of concurrency according to a newly defined criterion called the degree of fair concurrency. We also study the waiting time of our algorithm.

Organization. The rest of the paper is organized as follows. In Section 2, we present the preliminary concepts. Section 3 is dedicated to definitions of Maximal Concurrency and Fairness in committee coordination. Then, in Section 4, we propose our first snap-stabilizing algorithm that satisfies both Maximal Concurrency and 2-phase Discussion. In Section 5, we present our snap-stabilizing algorithm that satisfies Fairness and 2-phase Discussion. Our analysis on level of concurrency and waiting time is also presented in this section. Related work is discussed in Section 6. Finally, we present concluding remarks and discuss future work in Section 7.

2. Background

2.1. Distributed systems as hypergraphs

Considering the committee coordination problem in the context of distributed systems, professors and committees are mapped onto *processes* and *synchronization events* (*e.g.*, rendezvous) respectively. We assume that each process has a unique identifier and the set of all identifiers is a total order. We simply denote the identifier of a process p by p.

For the sake of simplicity, we assume that each committee has at least two members.[1] Hence, we model a *distributed system* as a simple self-loopless hypergraph $\mathcal{H} = (V, \mathcal{E})$ where V is a finite set of vertices representing processes and $\mathcal{E}$ is a finite set of *hyperedges*

[1] Adapting our results to take singleton committees into account is straightforward.