J. Parallel Distrib. Comput. 84 (2015) 94–107

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

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

On the competitiveness of scheduling dynamically injected tasks on processes prone to crashes and restarts

Chryssis Georgiou [a,*], Dariusz R. Kowalski [b]

[a] *Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus*
[b] *Department of Computer Science, University of Liverpool, Liverpool L69 3BX, United Kingdom*

HIGHLIGHTS

- New framework for fault-tolerant job scheduling with dynamic task arrivals.
- Centralized and distributed settings considered.
- Positive results: Competitive (efficient) algorithms wrt the number of pending jobs.
- Negative results: Conditions for non-competitiveness.

ARTICLE INFO

Article history:
Received 3 October 2013
Received in revised form
14 July 2015
Accepted 17 July 2015
Available online 26 July 2015

Keywords:
Task execution
Dynamic task injection
Processor crashes and restarts
Competitive analysis
Distributed algorithms

ABSTRACT

To identify the tradeoffs between efficiency and fault-tolerance in dynamic cooperative computing, we initiate the study of a task performing problem under dynamic processes' crashes/restarts and task injections. The system consists of n message-passing processes which, subject to dynamic crashes and restarts, cooperate in performing tasks that are continuously and dynamically injected to the system. Tasks are not known a priori to the processes. This problem abstracts todays Internet-based computations, such as Grid computing and cloud services, where tasks are generated dynamically and different tasks may become known to different processes. We measure performance in terms of the number of pending tasks, and as such it can be directly compared with the optimum number obtained under the same crash–restart–injection pattern by the best off-line algorithm. Hence, we view the problem as an online problem and we pursue competitive analysis. We propose several deterministic algorithmic solutions to the considered problem under different information models and correctness criteria, and we argue that their performance is close to the best possible offline solutions. We also prove negative results that open interesting research directions.

1. Introduction

Motivation. One of the fundamental problems in distributed computing is to have a collection of processes to collaborate in performing large sets of tasks. For such distributed collaboration to be effective it must be designed to cope with dynamic perturbations that occur in the computation medium (e.g., processes or communication failures). For this purpose, a vast amount of research has been dedicated over the last two decades in developing fault-tolerant algorithmic solutions and frameworks for various versions of such cooperation problems (e.g., [11,12,14,21–23,28,30]) and in deploying distributed collaborative systems and applications (e.g., [2,4,15,25,27]).

In order to identify the tradeoffs between efficiency and fault-tolerance in distributed cooperative computing, much research was devoted in studying the abstract problem of using n processes to cooperatively perform m independent tasks in the presence of failures (see for example [13,21,24]). In this problem, known as *Do-All*, the number of tasks m is assumed to be fixed and known a priori to all processes. Although there are several applications in which tasks can be known a priori, in todays typical Internet-based computations, such as Grid computing (e.g., [15]), Cloud services (e.g., [2]), and master–worker computing (e.g., [25,27]), tasks are generated dynamically and different tasks may become known to different processes. As such computations are becoming very popular, there is a corresponding need to develop efficient and

* Corresponding author.
E-mail addresses: chryssis@cs.ucy.ac.cy (C. Georgiou), D.Kowalski@liverpool.ac.uk (D.R. Kowalski).

http://dx.doi.org/10.1016/j.jpdc.2015.07.007
0743-7315/© 2015 Elsevier Inc. All rights reserved.

fault-tolerant algorithmic solutions that would also be able to cope with dynamic tasks injections.

Our contributions. In this work, in an attempt to identify the tradeoffs between efficiency and fault-tolerance in dynamic cooperative computing, we *initiate* the study of a task performing problem in which n message-passing processes, subject to dynamic crashes and restarts, cooperate in performing independent tasks that are continuously and dynamically injected to the system. Our investigation is based on a simple model of computation that abstracts key attributes, such as dynamic task arrivals, worst case occurrences of processes crashes and restarts, and level of information given to the processes. Our goal is to provide a rigorous analysis of the efficiency of algorithmic solutions – i.e., provide *provable* efficiency and fault-tolerance guarantees – and identify limitations even under the simple model (lower bound and impossibility results that are valid also in more complex settings). We believe that our investigation provides new insights on the complexity and fault-tolerance of dynamic task computations, on which follow up works could build on, either by enhancing the model to bypass the limitations or focus on extending our positive results to different (more complex) settings.

Basic framework: The computation is broken into synchronous rounds, in which each process is allocated tasks, receives messages sent to it in the prior round, performs local computations (including performing at most one task), and sends messages (if any). Unless otherwise stated, we assume that tasks are of unit-length, that is, it takes one round for a process to perform a task. This abstracts the situations where tasks consume comparable resources[1] and processors are homogeneous (hence one can specify the notion of a unit-length based on the tasks' requirements and processes' capabilities). An execution of an algorithm is specified under a *crash–restart–injection* pattern, that is, a collection of crash, restart and injection events; in a crash event a process crashes, in a restart event a crashed process restarts, and in an injection event, a task in injected in the system (the task is allocated to one or many processes). Then, the efficiency of an algorithm is measured in terms of the *maximum number of pending tasks* at the beginning of a round of an execution, taken over all rounds and all executions. This enables us to view the problem as an online problem and pursue *competitive analysis* [32], that is, compare the efficiency of a given algorithm with the efficiency of the best offline algorithm that knows a priori the crash–restart–injection patterns; we refer to the efficiency of the offline algorithm as OPT. More precisely, we say that an algorithm has $\mathrm{OPT} + x$ *pending-tasks competitiveness*, if the algorithm's maximum number of pending tasks (over all rounds and executions) is greater than OPT by at most an additive real number x. (A formal definition is given in Section 2.) Observe that the comparison to OPT means in fact that we compare our algorithm's efficiency to the efficiency of *all* other possible solutions, for any crash–restart–injection pattern.

Task performance guarantees: We consider two versions of the problem with respect to the task performance guarantees required by algorithmic solutions. The first one, which constitutes the basic *correctness* property, requires that no task is lost, that is, a task is either performed or the information of the task remains in the system. The second and stronger property, which we call *fairness*, requires that all tasks injected in the system are eventually performed. As we mention below, we draw a line on the conditions under which these two properties can be satisfied and with what cost.

[1] We refer the reader to [21] for a discussion on cooperative applications involving tasks that are independent and consume comparable resources.

Our approach: We deploy an *incremental* approach in studying the problem. We first assume that there is a centralized authority, called *central scheduler*, that at the beginning of each round informs the processes (that are currently operational) about the tasks that are still pending to be performed, including any new tasks injected to the system in this round. The reason to begin with this assumption is two-fold: (a) The fact that processes have consistent information on the number of pending tasks enables us to focus on identifying the inherent limitations of the problem under processes failures/restarts and dynamic injection of tasks without having to implement information sharing amongst processes. The algorithmic solutions developed under this *information* model are used as building blocks in versions of the problem that deploy weaker information models. Furthermore, lower bound results developed in this information model are also valid for weaker information models. (b) Studying the problem under this assumption has its own independent interest, as the central scheduler can be viewed as an *abstraction* of a monitor used for monitoring the computation progress and providing feedback to the computing elements. For example it could be viewed as a *master server* in Master–Worker Internet-based computations such as SETI [25] or Pregel [27], or as a *resource broker/scheduler* in Computational Grids such as EGEE [15].

We then limit the information provided to the processes. We consider a weaker centralized authority, called *central injector*, which informs processes, at the beginning of each round, only about the tasks injected to the system in this round and information about which tasks have been performed only in the previous round. We show how to transform solutions for the task performing problem under the model of central scheduler into solutions for the problem under the model of central injector with the expense of sending a quadratic number of messages in every round. It also occurs that a quadratic number of messages must be sent in some rounds by any correct distributed solution for the considered problem in the model of central injector.

With the gained knowledge and understanding, we then show how processes can obtain common knowledge on the set of pending tasks without the use of a centralized authority. We now assume the existence of a *local injector* that allocates tasks to processes without giving them any global information (for example, each process may be allocated tasks that no other process in the system has been allocated, or only a subset of processes may be allocated the same task). The injector can be viewed, for example, as a local daemon of a distributed application that provides local information to the process that is running on. We show that solutions to this more general setting come with minimal cost to the competitiveness, provided that *reliable multicast* [7] is available.

Our results: We now summarize our results. (All results concern deterministic solutions.)

(a) Limitations on competitiveness: We first show a lower bound of $\mathrm{OPT} + n/3$ on the pending-tasks competitiveness of any deterministic algorithm, even for algorithms that make use of messages and are designed for restricted forms of crash–restarts patterns (cf. Section 3). The lower bound is proved for the model of central scheduler, but since this is the strongest information model, the result holds also for the other two weaker information models.

(b) Solutions guaranteeing correctness: Within the model of central scheduler we develop the deterministic algorithm AlgCS that does not make any use of message exchange amongst processes and achieves $\mathrm{OPT} + 2n$ pending-tasks competitiveness; in view of the lower bound above, the algorithm is optimal within a constant factor on the additive term of the competitiveness formula. Using a *generic transformation* we obtain algorithm AlgCI for the model with central injector with