Future Generation Computer Systems 26 (2010) 847–856

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Characterizing fault tolerance in genetic programming

Daniel Lombraña González [a,*], Francisco Fernández de Vega [a], Henri Casanova [b]

[a] University of Extremadura, Santa Teresa Jornet 38, 06800 Mérida, Badajoz, Spain

[b] University of Hawai'i at Manoa, PST #317, 1680 East-West Road, Honolulu, HI 96822, USA

ARTICLE INFO

Article history:
Received 7 July 2009
Received in revised form
1 February 2010
Accepted 10 February 2010
Available online 18 February 2010

Keywords:
Fault tolerance
Parallel genetic programming
Desktop grids

ABSTRACT

Evolutionary algorithms, including genetic programming (GP), are frequently employed to solve difficult real-life problems, which can require up to days or months of computation. An approach for reducing the time-to-solution is to use parallel computing on distributed platforms. Large platforms such as these are prone to failures, which can even be commonplace events rather than rare occurrences. Thus, fault tolerance and recovery techniques are typically necessary. The aim of this article is to show the inherent ability of parallel GP to tolerate failures in distributed platforms without using any fault-tolerant technique. This ability is quantified via simulation experiments performed using failure traces from real-world distributed platforms, namely, desktop grids, for two well-known problems.

1. Introduction

Evolutionary algorithms (EAs) are employed to solve real-world problems. However, when difficult problems are faced, large and often prohibitive times-to-solution ensue. A common approach is then to use parallel versions of the algorithm. Successful parallel evolutionary algorithms (PEAs) have been proposed [1–3]. Additionally, development frameworks for PEAs have been implemented (e.g., Calypso [4]), some specifically for parallel genetic programming (PGP) [5].

PEAs can be run on potentially large-scale parallel computing platforms. Two types of parallel platforms have achieved very high scales: *clusters* and *desktop grids*. In the last decade large clusters have become mainstream, and clusters account today for more than 80% of the Top500 list, which ranks the 500 fastest supercomputers based on the LINPACK benchmark [6]. The term "desktop grid" (DG) refers to distributed networks of heterogeneous individual systems that contribute computing resources when idle. A well-known example of a DG is the Berkeley Open Infrastructure for Network Computing (BOINC) [7], which hosts the famous SETI@home [8] project. At the time this article is being written, BOINC enlists around 500,000 volunteer computers. Smaller, but still impressive, DG infrastructures, are deployed within enterprises or data centers [9]. Such deployments comprise more powerful, more available, and less heterogeneous computers than Internet-wide DGs. The main advantage of DGs is that they provide large-scale parallel computing capabilities at a very low cost for specific types of application.

The aforementioned large-scale parallel computing platforms hold promises for running large PEAs. However, with large scale there is a higher risk that processors experience failures during the execution of an application (e.g., a crash). In this paper the terms "failure" and "fault" are used without making the subtle distinction between them, as it is not necessary for our purpose. Failures occur frequently in large-scale clusters [10]. In DGs, failures are the common case: a participating computer can be reclaimed by its owner at any time (e.g., when the owner launches an application, when the keyboard/mouse is used). In this case, the DG application is abruptly suspended or terminated, which it is seen as a failure.

In order to circumvent and/or alleviate failures, many researchers have developed different techniques for an application to not be terminated when one or more of the participating processors experience a failure. This ability is known as *fault tolerance*, and it ensures that the application behaves in a well-defined manner (e.g., with graceful degradation of performance) when a failure occurs [11]. Various fault tolerance techniques have been developed [12]. These techniques can be employed with parallel applications, and support many types of computational and communication failures [13]. In general, the employment of fault tolerance mechanisms requires the modification of the application, and sometimes the parallel algorithms themselves. The developer thus could face a sharp increase in software complexity. For this reason, generic fault tolerance solutions have been developed as libraries or software environments [14–16].

To the best of our knowledge, there has been little investigation of the behaviors of PEAs in general, and of PGP in particular, in the presence of failures. Nevertheless, there are different tools

* Corresponding author.
E-mail addresses: daniellg@unex.es (D. Lombraña González), fcofdez@unex.es (F. Fernández de Vega), heric@hawaii.edu (H. Casanova).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.02.006

available that can be used to parallelize and run any EA, and thus GP, in volunteer computing environments [17], where failures are common.

In previous work [18,19], we presented preliminary results about fault tolerance in PGP under several simplified assumptions. We showed that PGP applications exhibit inherent fault-tolerant behaviors. Therefore, it seems feasible to run them on large-scale computing infrastructures, which suffer from failures, but without the burden of implementing/using any kind of fault tolerance techniques, and without sacrificing overall application efficiency significantly. We then extended those preliminary results in two ways [20]. First, two different PGP problems were used for running simulations using host availability data collected from real-world DG deployments (instead of using simplistic, and ultimately unrealistic, processor availability models). Second, the simulations used availability data from different platforms, making it possible to study the impact of different host availability profiles on application execution. To the best of our knowledge, this was the first time that an attempt for characterizing PGP had been performed from the fault tolerance point of view. The results showed that, in some specific contexts, PGP can tolerate various failure rates.

All previous results were obtained using a stringent assumption: once unavailable, a resource never becomes available again. This is, however, not the case in real-world DGs. In this paper we extend the work in [20] and run simulations in which resources can become available again. This should further improve the *graceful degradation* feature of PGP as the number of resources fluctuates throughout application execution instead of continuously decreasing.

This paper is organized as follows. Section 2 reviews related works beyond the ones described earlier. Section 3 provides an overview of the different types of failure that may arise as well as the relevant fault tolerance techniques. Section 4 describes our experimental methodology, and results are discussed in Section 5. Section 6 concludes the paper with a summary of our results and future directions.

2. Background and related work

When using EAs, and especially GP, to solve real-world problems, researchers and practitioners often face prohibitively long times-to-solution on a single computer. For instance, Trujillo et al. required more than 24 h to solve a computer vision problem [21], and times-to-solution can be much longer, measured in weeks or even months. Consequently, several researchers have studied the application of parallel computing to spatially structured EAs in order to shorten times-to-solution [1–3]. Such PEAs have been used for decades, for instance, on the Transputer platform [22], or, more recently, via software frameworks such as BEAGLE [23], grid-based tools like Paradiseo [24], or BOINC-based EA frameworks for execution on DGs [17].

Failures in a distributed system can be local, affecting only a single processor, or they can be communication failures, affecting a large number of participating processors. Such failures can disrupt a running application, for instance mandating that the application be restarted from scratch. As distributed computing platforms become larger and/or lower cost through the use of less reliable or non-dedicated hardware, failures occur with higher probability [25–27]. Failures are, in fact, the common case in DGs. For this reason, fault-tolerant techniques are necessary so that parallel applications in general, and in our case PEAs, can benefit from large-scale distributed computing platforms. Failures can be alleviated, and in some cases completely circumvented, using techniques such as checkpointing [28], redundancy [29], long-term memory [30], specific solutions to message passing [31] or rejuvenation frameworks [32]. It is necessary to embed the techniques in the application and the algorithms. While some of these techniques may be straightforward to implement (e.g., failure detection and restart from scratch), the more involved ones typically lead to an increase in software complexity. Regardless, fault tolerance techniques always requires extra computing resources and/or time.

Currently available PEA frameworks employ fault-tolerant mechanisms to tolerate failures in distributed systems like in DGs: for instance, ECJ [33], ParadisEO [34], DREAM [35] or Distributed BEAGLE [23]. These frameworks have distinct features (programming language, parallelism models, etc.) that may be considered in combination with DGs, and provide different techniques to cope with failures.

- ECJ [33] is a Java framework that employs a master–worker scheme to run PEAs using TCP/IP sockets. When a remote worker fails, ECJ handles this failure by rescheduling and restarting the computation to another available worker.
- ParadisEO [34] is a C++ framework for running a master–worker model using MPI [36], PVM [37], or POSIX threads. Initially, ParadisEO did not provide any fault tolerance. The developers later implemented a new version on top of the Condor-PVM resource manager [38] in order to provide a checkpointing feature [28]. This framework, however, is not the best choice for DGs because these systems are: (i) loosely coupled and (ii) workers may be behind proxies, firewalls, etc., making it difficult to deploy a ParadisEO system.
- DREAM [35] is a Java peer-to-peer (P2P) framework for PEAs that provides a fault tolerance mechanism called *long-term memory* [30]. This framework is designed specifically for P2P systems. As a result, it cannot be compared directly with our work since we focus on a master–worker architecture on DGs.
- Distributed BEAGLE [23] is a C++ framework that implements the master–worker model using TCP/IP sockets as ECJ. Fault tolerance is provided via a simple time-out mechanism: a computation is re-sent to one or more new available workers if this computation has not been completed by its assigned worker after a specified deadline.

While these PEA frameworks provide fault-tolerant features, the relationship between fault tolerance and specific features of PEAs has not been studied.

An interesting and relatively recent development is the use of *dynamic populations* [39–42], i.e., the population size is reduced and/or increased in order to minimize fitness stagnation. In 2003, Fernández et al. [40] introduced a new operator named *plague*, which reduces the population size at a linear rate. The same year, Luke et al. [41] introduced a similar idea: reducing the population size according to different *layouts*. In 2004, Tomassini et al. [39] presented a study on dynamic populations by removing/adding individuals to avoid fitness stagnation. More recently, Kouchakpour et al. [43] introduced a population variation scheme to add/remove individuals from the population, and in [42] improved the work presented by Tomassini et al. by proposing a new pivot function and four new measures for characterizing the stagnation phase.

Our key observation is that the loss of individuals due to worker failures inherently leads to dynamic populations, albeit in a random and thus less controlled fashion. Based on this observation, the intriguing question is whether one could simply allow individuals to be lost without taking any particular fault-tolerant measures, thereby achieving a dynamic population and fault tolerance "for free". We first explored this idea in [44,45], and, as explained in Section 1, attempted a preliminary quantification of the fault tolerance characteristics of PGP in [18,19]. These results were extended in our more recent work [20], using more realistic failure models based on real-world observations of DG