Applied Soft Computing 24 (2014) 1095–1104

Contents lists available at ScienceDirect

Applied Soft Computing

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

Monitoring applications: An immune inspired algorithm for software-fault detection

Rui Ligeiro [a,b]

[a] INOV INESC – Instituto de Novas Tecnologias, Rua Alves Redol 9, 1000-029 Lisboa, Portugal
[b] CMAF – Instituto Investigação Interdisciplinar, Univ. Lisboa, Av. Gama Pinto 2, 1649-003 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 24 February 2013
Received in revised form 11 May 2014
Accepted 11 August 2014
Available online 1 September 2014

Keywords:
Artificial immune system
Fault detection
Fault injection
Reinforcement learning
Monitoring
Metrics

ABSTRACT

Large-scale software systems are in general difficult to manage and monitor. In many cases, these systems display unexpected behavior, especially after being updated or when changes occur in their environment (operating system upgrades or hardware migrations, to name a few). Therefore, to handle a changing environment, it is desirable to base fault detection and performance monitoring on self-adaptive techniques.

Several studies have been carried out in the past which, inspired on the immune system, aim at solving complex technological problems. Among them, anomaly detection, pattern recognition, system security and data mining are problems that have been addressed in this framework.

There are similarities between the software fault detection problem and the identification of the pathogens that are found in natural immune systems. Being inspired by vaccination and negative and clonal selection observed in these systems, we developed an effective self-adaptive model to monitor software applications analyzing the metrics of system resources.

1. Introduction

Large-scale software systems are difficult to manage and monitor. These systems may display unexpected behavior, especially after being updated or when changes occur in their environment (operating system upgrades or hardware migrations, to name a few). Novel situations require robust defensive mechanisms, but monitoring can be rather time-consuming, requiring many efforts and tools [1–6] for the purpose.

A failure or malfunction occurs when the system behavior deviates from its initial specification, this being usually associated to the detection of a system error. In case of a failure in an application component, it must be detected quickly and, preferably, the overall system should be kept working, even with limitations. CPU usage, memory usage, load average and thread count are, among others, useful resource indicators of the efficiency and performance of a system. These metrics are highly correlated with the characteristics of the host where the software applications execute and, in general, when something wrong happens, some of the metrics reach values outside their usual ranges. Thus, the creation of an adaptive monitoring mechanism that ensures system fault detections based on resource metrics is a natural choice.

Despite all recent biological findings, lots of uncertainty still exists regarding how Nature works. However, in these last decades, biological systems have been used as a source of inspiration to solve complex technological problems, going far beyond the earlier boundaries of computer science. For example, analogies between the defending mechanisms of the immune system and anomaly detection in computer systems have been largely studied since 1994, after publications by Forrest et al. [8] and Kephart [9]. As a matter of fact, the vertebrate immune system has been the object of study by several authors [11–18], and as here, we give especial relevance to the insights most pertinent to the monitoring model, particularly the self/non-self discrimination, vaccination and some specific aspects of the adaptive immune response. The vertebrate immune system it is a complex system composed of a large collection of cells with several defense mechanisms that protect the body against diseases by recognizing, attacking and destroying pathogens. The system is divided into two inter-related branches: the *innate immune system* and the *adaptive immune system*. Roughly speaking, the innate immune system acts very quickly to the first signs of infection, being crucial to the initial inflammatory response by recognizing and signaling the adaptive immune response, whereas the adaptive immune system has the ability to change, improving the immune response during the lifetime of the organism. Note that the learning, memory and adaptation capabilities of the adaptive immune system emerge without any central control.

Lymphocytes, a special type of white blood cell with the function to recognize "non-self" antigens, are the most important agents of the adaptive immune system. B-cells and T-cells are the two main types of lymphocytes that together recognize and kill antigens. While B-cells produce and release large amounts of antibodies that attack pathogens, T-cells orchestrate the response of other cells as well as directly induce the death of cells that show signs of having been invaded by pathogens. The lymphocytes surface is covered with receptors that identify antigens by partial matching its shape. Consider, for instance, receptors and antigens as two pieces of LEGO, that even if they don't exactly join together, there are some complementary parts between each other. *Affinity* is the term used for the degree of recognition of antigens by lymphocyte receptors – stronger recognition corresponds to higher affinity and vice versa.

T-cells mature in the thymus gland, an organ located in the upper region of the chest to which T-cells travel after being created by the bone marrow in immature form. In this organ a process takes place called *negative selection*, responsible for eliminating T-cells capable of attacking the body's own cells. Nevertheless, this discrimination (*self/non-self*) can fail, resulting in the development of autoimmune diseases.

E-mail addresses: rui.ligeiro@inov.pt, rmligeiro@fc.ul.pt, rui.ligeiro@gmail.com

http://dx.doi.org/10.1016/j.asoc.2014.08.021
1568-4946/

The reproduction of lymphocytes is based in a principle known as *clonal selection*. Once activated, B-cells produce and segregate antibodies, proliferating in a quantity proportional to the degree of affinity with the antigen. B-cells are also stimulated by T-cells (T-helper cells) to divide into offspring cells, which are very similar to their parent. This preferential proliferation of the most capable cells has clear similarities with Darwin's evolution principle. The immune system maintains a population of long-lived memory cells after clearance of infection and recruits newly generated B-cells into the memory. *Immune memory* enables the immune system to act quickly and efficiently in protecting the body in case it is infected by similar pathogens in the future. *Vaccination* follows the same principle. Summarizing, the essential features of the natural immune systems are distributed detection, self-organization, multi-layer structure, diversity, autonomy, imperfect detection, learning, memory and adaptability.

Several immune inspired algorithms have been developed and a relatively new research field, called Artificial Immune Systems (AIS), arose as a new computing paradigm. Nevertheless, AIS has not been used so far, to our knowledge, as a tool to monitor concrete software applications using resource usage metrics. Our model, for monitoring software applications, is based on metrics of the system resources and inspired in the natural mechanisms of the immune system, briefly discussed above.

There are similarities between the software fault detection problem and the identification of pathogens in natural immune systems. It should be mentioned that the natural immune system is merely used as a metaphor for anomaly detection and we are not trying to imitate all its features and detailed operation. Our inspiration comes mostly from the following three features:

1.1. Self/non-self discrimination

A healthy immune system is able to differentiate between the cells of its organism, know as *self*, and the foreign elements (antigens) that attack the organism, know as *non-self*. In the same way, an anomaly points toward a deviant behavior in relation to what is expected and characteristic of the system. Thus, inspired by the vertebrate adaptive immunity response, an algorithm is developed to distinguish common behavior of the host (*self*) from faults (*non-self*) in software applications.

1.2. Vaccination

The reason why we do not acquire some diseases more than once is because the immune system remembers pathogens. Vaccination is a good evidence that the immune system has memory. It consists in introducing into an organism some harmless organisms, which provoke an immune response against the foreign elements. As a consequence, immunological memory is induced which enables the immune system to act quickly and efficiently in protecting the body when it is actually infected by the real pathogens at some future time. Having this in mind, a kind of fault injection learning mechanism was created, as a part of the monitoring model. A fault injection is an application of an artificial malfunction, inserted into a particular monitored system with the purpose of simulating a specific error. Note that the monitored system is not actually affected by the fault. This process occurs in the interface between the monitoring model and the monitored application. It consists in intercepting metrics collected from the monitoring model into the monitored application and changing their values to outside the normal range.

1.3. Adaptive immune response

The adaptive immune system is composed of a large collection of cells with no central control, which together have the ability to improve the immune response during the lifetime of the host. The system evolves based on the principles of mutation and selection, producing a number of lymphocytes proportional to the degree of binding (affinity) with the antigen. As stated above, in software systems, unexpected behavior requires robust adaptive defensive mechanisms capable of recognizing new faults. As proposed by de Castro and Von Zuben [10], one is here inspired by the clonal selection concept, together with the affinity maturation process of the immune response, to create an adaptive monitoring mechanism.

In this paper we show that not only the model performs well in detecting faults, but also fault injection and reinforcement learning substantially decrease the detection of false positives. The article begins by reviewing the most important aspects of software-fault monitoring, describes the faults that are simulated as well as the identification of all the metrics that are collected in the monitored system. In Section 3, we present a computational algorithm that, in addition to detecting anomalies, also identifies its type. After that, the results and discussion of the simulation are presented as well as the most relevant conclusions.

2. Preliminary knowledge

We advocate the use of monitoring as a major design principle to increase safety, reliability and dependability of software applications. Many tools have been proposed for runtime monitoring with the purpose of detecting, diagnosing and recovering from software faults. Nelly Delgado and colleagues described the taxonomy of software-fault monitoring systems and presented a state-of-the-art of the tools used to detect faults (for details, see [6] and references therein). Note that none of the tools referred in their study are based on metrics of the system resources together with immune system inspiration to distinguish the common behavior of the host (*self*) from faults (*non-self*).

In 1994 Forrest et al. applied to the problem of computer viruses (see [8] for details), is of paramount importance for the scientific community, because it unifies a wide variety of computational situations by treating them as the problem of distinguishing self from non-self. Later on, enhancements were made to the original version of the Negative Selection algorithm proposed by Forrest et al. ([20–23], to name a few), but the main features remained unchanged. Another valuable application of AIS-based algorithms for fault prediction is the study by Catal and Diri [35]. The authors analyzed the performance of several existing classifiers using a different kind of metrics: software metrics (method-level and class-level; see [7] for details on software fault prediction metrics). When compared to others, AIS algorithms present remarkable results in predicting faults, however no experimentation are presented in detecting faults in real time, like we do here. Two other relevant methods address the fault detection problem using AIS combined with other technics. One is an approach based on conventional fuzzy soft clustering and AIS for multiple sensor data fusion and fault detection [36], the other is a multi-objective AIS to optimize parameters of a Support Vector Machine (SVM) applied to fault diagnosis of induction motors and anomaly detection problems [37]. Our model mimics a system that has features very close to a real system in contrast to the mentioned works that although showing good results, seem to be not sufficiently mature to face the requirements and complex behavior of practical applications.

To the best of the author's knowledge, the only work presenting an evolutionary technique that also invokes a set of resource usage metrics for software faults detection is that by Wong et al. [24]. Due to this similarity, their approach will be discussed in more detail in the Section 4.

In general terms, despite of all the important studies carried out in software monitoring, we did not find one that detects faults using metrics together with artificial immune adaptation technics. Furthermore, most of them are based in disparate approaches and methodologies as the existence of an oracle, i.e., determining if the systems behavior under test is or not acceptable, or concern whether the design or implementation of the system meets the requirements (or specifications) or the instrumentation of a program code.

The main goal of runtime software-fault monitoring is to observe the software behavior in order to determine whether it complies with its intended purpose, in other words, to determine if it is consistent with a given specification [19]. Avizienis and Laprie [25] gave widely accepted definitions of systems fault, error and failure. In summary, a system *failure* occurs when the delivered service deviates from the required service because the system was erroneous: an *error* is that part of the system state that is liable to lead to a *fault* in the system. A fault is active when it causes an error and results in an incorrect state that may or may not lead to a failure. Some faults are, deliberately or not caused by humans, whereas others are trigged by natural phenomena without human participation. Both may cause a huge impact, affecting partially or even completely the integrity of the whole system.

Here, we do not need to treat differently faults, errors or failures. We simply use the term *fault*, considered as a malfunction that affects the proper functioning or full availability of a system, leading some particular resource metrics to reach values outside their usual ranges. Three faults were simulated to evaluate the performance of our model (which was developed in Java):