The Journal of Systems and Software 000 (2016) 1–18

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Improving software performance and reliability in a distributed and concurrent environment with an architecture-based self-adaptive framework

Chung-Horng Lung*, Xu Zhang, Pragash Rajeswaran

Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada

ARTICLE INFO

Article history:
Received 22 December 2014
Revised 7 June 2016
Accepted 29 June 2016
Available online xxx

Keywords:
Autonomic computing
Software architecture
Performance
Reliability
Patterns
Distributed and concurrent architecture
Elastic computing

ABSTRACT

More and more, modern software systems in a distributed and parallel environment are becoming highly complex and difficult to manage. A self-adaptive approach that integrates monitoring, analyzing, and actuation functionalities has the potential to accommodate an ever dynamically changing environment. This paper proposes an architecture-level self-adaptive framework with the aim of improving performance and reliability. To meet such a goal, this paper presents a Self-Adaptive Framework for Concurrency Architectures (SAFCA) that consists of multiple well-documented architectural patterns in addition to monitoring and adaptive capabilities. With this framework, a system using an architectural alternative can activate another alternative at runtime to cope with increasing demands or to recover from failure. Five adaptation mechanisms have been developed for concept demonstration and evaluation; four focus on performance improvement and one deals with failover and reliability enhancement. We have performed a number of experiments with this framework. The experimental results demonstrate that the proposed adaptive framework can mitigate the over-provisioning method commonly used in practice. As a result, resource usage becomes more efficient for most normal conditions, while the system is still able to effectively handle bursty or growing demands using an adaptive mechanism. The performance of SAFCA is also better than systems using only standalone architectural alternatives without an adaptation scheme. Moreover, the experimental results show that a fast recovery can be realized in the case of failure by conducting an architecture switchover to maintain the desired service.

1. Introduction

The complexity of computing systems and variety of computing devices have risen dramatically (EMA, 2006). As a result, a significant portion of system management is closely tied to configuration, the process of assembling and/or adjusting components towards a desired solution (Keller et al., 2007). Human-driven configuration operations have been widely used at various stages in practice, and constitute the main factor behind high operational costs and errors (Kephart and Chess, 2003).

Take cloud computing as one example of such configuration complexity and system management. The demand for cloud datacenters is highly dynamic. Under normal workload conditions, cloud servers are for the most part more than adequate to handle traffic demands. In fact, low utilization of datacenters has often been observed (e.g., 6%, 7%, 12%, 20%–40% have been reported for various datacenters) (Benik and Ventures, 2013). However, the server may still become unavailable due to unprecedented demands (Welsh and Culler, 2003). To deal with such diverse demands, many systems are typically configured or over-provisioned based on peak demands. The primary problem with static over-provisioning is poor resource efficiency and the associated high cost of non-peak periods.

Autonomic computing has been proposed to provide solutions that help manage resources more effectively, i.e., not only scaling up but also scaling down of system resources in response to dynamic demands. In other words, autonomic computing supports system elasticity so as to allocate or de-allocate resources dynamically due to changing workloads and possibly other factors as well. Manually configuring systems for dynamic scenarios has been shown to be inadequate (IBM, 2006). Rather, self-adaptive solutions are better suited for such problems.

Specifically, the objectives of autonomic computing are to enable a system to automatically configure, heal, protect, and opti-

* Corresponding author.
E-mail addresses: chlung@sec.carleton.ca (C.-H. Lung), zhangxu@sec.carleton.ca (X. Zhang), pragasra@gmail.com (P. Rajeswaran).

http://dx.doi.org/10.1016/j.jss.2016.06.102
0164-1212/

mize itself (Kephart and Chess, 2003). A number of organizations have been working on projects related to autonomic computing (Muller et al., 2006), particularly in the area of cloud computing for distributed and concurrent applications due to the diverse demands and elevated system complexity. Hence, the development of self-adaptive or self-managing systems has become a crucial topic within the software engineering community (Huebscher and McCann, 2008; SEAMS, 2016).

Self-adaptation can be realized at various levels of abstraction. To deal with the increasing complexity of software systems, this paper advocates an architecture-based self-adaptive approach. Garlan et al. (2009) pointed out that self-adaptation at the architectural level has enormous potentials. First, the rich set of knowledge in software architecture analysis (ATAM, 2000; Lung and Kalaichelvan, 2000) facilitates system design for runtime adaptation. Next, the abstract architecture description reveals the key system-level properties and expected behaviors. Further, the architectural model exposes explicit system integrity constraints, thereby supporting validity check.

The main objective of this paper is to take software architecture analysis (e.g., ATAM, 2000; Lung and Kalaichelvan, 2000) one step further by utilizing a self-adaptive scheme built on the strengths of different architectural alternatives (Garlan et al., 2009). Traditional software architecture analysis methods have investigated tradeoffs among architectural alternatives based on scenarios and non-functional attributes. However, the target system is typically built solely on one selected architectural alternative. On the other hand, each architectural option may have its own properties and strengths for different scenarios. Using our proposed approach, multiple architectural alternatives can be selected after analysis before being integrated into a framework. More importantly, an adaptive policy is constructed and built into the framework itself based on the strengths of each architectural alternative so as to support self-adaptation for different scenarios.

In other words, the framework can dynamically facilitate architecture-level adaptation at runtime, depending on the strengths of selected architectural alternatives, the pre-established adaptive policy, and monitored runtime observations. The target problem domain of this paper is the distributed and concurrent systems, while the main non-functional attributes considered are performance and reliability. We have built such a self-adaptive framework by making use of well-known architectural patterns in this problem domain (Schmidt et al., 2000), as well as our own experimental observations for those architectural patterns (Lung et al., 2014; Zhang and Lung, 2010).

The primary contribution of the paper is twofold:

(1) This paper presents a novel approach to supporting architecture-level adaptation. The approach extends traditional software architecture analysis and devises multiple adaptive policies that make use of the strengths of different architectural alternatives to enhance performance and reliability.
(2) We have developed such a framework in the area of distributed and concurrent processing. Further, we have conducted thorough experiments to validate the proposed approach with multiple adaptation mechanisms. The results from a number of experiments demonstrate the feasibility of architecture-based adaptation at runtime and subsequent improvement of performance and reliability.

This paper is organized as follows: Section 2 briefly describes the background to autonomic computing and concurrency architectural patterns (or alternatives that have been used in our studies). Section 3 describes the self-adaptive framework, SAFCA, including different policies for increasing performance and one policy for improving reliability. Section 4 illustrates the experiments and results. Section 5 discusses the threats to validity. Section 6 presents related work on self-adaptive systems. Finally, Section 7 presents our conclusions and a look at future research directions.

2. Background

This section briefly presents the related background to autonomic computing and various existing concurrency architectural patterns or alternatives to thread management.

2.1. Overview of autonomic computing

A particular challenge of modern computing systems is their elevated level of complexity. The manual configuration and control of a complex computing system is both time-consuming and error-prone. Moreover, those issues become more serious as the size of those systems tends to increase rapidly. The main objective of autonomic computing is to define the rules for a system to control its own behavior, such that the system itself can dynamically manage its own actions to accommodate changes or demands (EMA, 2006; Huebscher and McCann, 2008).

There are four main functional areas for autonomic computing, which may be highlighted as follows (IBM, 2006):

- Self-configuration: automatic configuration of system components;
- Self-healing: automatic discovery and correction of faults;
- Self-optimization: automatic monitoring and management of resources to achieve optimal solutions with respect to the requirements and monitored data;
- Self-protection: pro-active identification and protection from arbitrary attacks.

This paper focuses on self-optimization or self-adaptation for better performance, and self-healing for greater reliability.

An autonomic manager, used to manage software or hardware resources, is a key component of the autonomic system. The autonomic manager has four main functions (IBM, 2006):

- Monitor: collect data that the autonomic manager needs from the system;
- Analyze: analyze the data to determine if something needs to be changed;
- Plan: create a plan or sequence of actions that specifies the necessary changes;
- Execute: perform the actions according to the plan.

Autonomic computing has received a great deal of attention recently, primarily due to the increasing popularity of cloud computing. One key advantage of cloud computing is its flexibility for dynamic on-demand resource allocation. Various techniques in elastic computing (Galante and de Bona, 2012) or auto-scaling (Lorido-Botran et al., 2014) have been investigated to support dynamic resource allocation. Elastic solutions have four main characteristics: (1) scope, (2) policy, (3) purpose, and (4) method (Galante and de Bona, 2012). Lorido-Botran et al. (2014) also categorize existing auto-scaling approaches into five groups: threshold-based policies, reinforcement learning, queuing theory, control theory, and time-series analysis.

In summary, autonomic computing becomes essential as a result of the increased complexity of systems, uncertainty of the environment, and popularity of cloud computing. One crucial research issue is how to build capabilities into a system, allowing it to adapt its behavior in response to a dynamically changing environment.

Please cite this article as: C.-H. Lung et al., Improving software performance and reliability in a distributed and concurrent environment with an architecture-based self-adaptive framework, The Journal of Systems and Software (2016), http://dx.doi.org/10.1016/j.jss.2016.06.102