The Journal of Systems and Software 84 (2011) 2123–2138

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Evaluating the impacts of dynamic reconfiguration on the QoS of running systems

Wei Li*

Centre for Intelligent and Networked Systems, and School of Information & Communication Technology, Central Queensland University, Rockhampton, QLD 4702, Australia

ARTICLE INFO

Article history:
Received 27 June 2010
Received in revised form 18 May 2011
Accepted 25 May 2011
Available online 12 July 2011

Keywords:
Dynamic reconfiguration
QoS assurance
Software maintenance
Software evolution
Quantitative analysis

ABSTRACT

A major challenge in dynamic reconfiguration of a running system is to understand in advance the impact on the system's Quality of Service (QoS). For some systems, any unexpected change to QoS is unacceptable. In others, the possibility of dissatisfaction increases due to the impaired performance of the running system or unpredictable errors in the resulting system. In general it is difficult to choose a reasonable reconfiguration approach to satisfy a particular domain application. Our investigation on this issue for dynamic approaches is four-fold. First, we define a set of QoS characteristics to identify the evaluation criteria. Second, we design a set of abstract reconfiguration strategies bringing existing and new approaches into a unified evaluation context. Third, we design a reconfiguration benchmark to expose a rich set of QoS problems. Finally, we test the reconfiguration strategies against the benchmark and evaluate the test results. The analysis of acquired results helps to understand dynamic reconfiguration approaches in terms of their impact on the QoS of running systems and possible enhancements for newer QoS capability.

1. Introduction

Dynamic reconfiguration is a widely studied topic in the runtime evolution of software systems. One way to assist appropriate use of a particular dynamic reconfiguration approach is to understand the impact that the approach may have on a running system. This article conducts an evaluation with respect to *Quality of Service* (QoS), by which we refer to end-user observable performance attributes such as *throughput* and *response time* of a running system, and *QoS assurance*, by which we refer to the capability of a reconfiguration approach to maintain pre-determined QoS levels for a running system under reconfiguration (RSUR). The background and necessity of this research are as follows.

It is necessary for a software system to evolve continuously in order to remain useful (Kramer & Magee, 1990; Soria et al., 2009; Vandewoude et al., 2007). This evolution adapts the system to the changing business needs that diverge from the original requirements placed upon it (Morrison et al., 2007). The evolution comprises incremental upgrade of the functionality and modification of the topology of a software system (Fung & Low, 2009). In its lifecycle, a software system may evolve multiple times; *a reconfiguration* refers specifically to a specific process of reconfiguration at a specific point in the history of a system evolution.

The evolution of software systems can be accomplished by static or dynamic reconfiguration. *Static reconfiguration* involves stopping a running system, recompiling its binaries, rebuilding its data, altering its topology and then restarting the system. A static reconfiguration is performed out of band and is therefore not viable for the applications that are featured as mission-critical or uninterruptible (Vandewoude et al., 2007). For mission-critical services or 24/7 business services, such as those described by Warren et al. (2006), a running system cannot be reconfigured statically due to unacceptable down time. For other systems, down time reduces the productivity of users and increases costs for the businesses (Patterson, 2002).

Dynamic reconfiguration involves upgrading/altering a system's functionality and topology at runtime via addition, deletion and replacement of components and connections. Dynamic reconfiguration promotes service availability and self-adaptation because it is able to make reconfiguring or adaptive steps as a part of normal system execution.

From the early work of Kramer and Magee (1990) to the recent work of Morrison et al. (2007), Soria et al. (2008, 2009) and Vandewoude et al. (2007), dynamic reconfiguration has advanced significantly in modeling (Ajmani et al., 2006; Almeida et al., 2004; Dowling & Cahill, 2001a,b; Warren et al., 2006) and implementation (Evans, 2004; Janssens et al., 2007; Kim & Bohner, 2008; Rasche & Polze, 2008; Tewksbury et al., 2001; Truyen et al., 2008).

However, recently Vandewoude et al. (2007) re-investigated a typical dynamic approach (Kramer & Magee, 1990) and noted that it could result in significant disruption to the application being updated. That work raised the issue that impact evaluation is significant for dynamic reconfiguration.

Moreover, existing approaches have been evaluated individually, but not quantitatively compared in a unified evaluation context. As a result, the questions: *how is dynamic reconfiguration*

* Tel.: +61 7 4930 9686; fax: +61 7 4930 9729.
E-mail address: w.li@cqu.edu.au

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.05.060

different from static reconfiguration, and *how are dynamic approaches different from each other* remain open.

In this article we argue that addressing dynamic reconfiguration on a running system without systematic evaluation increases the possibility of user dissatisfaction, due to impaired performance or unpredictable errors. Furthermore, without comparison, it is difficult for a developer to choose the right dynamic approach to satisfy a particular domain application or enhance an available approach for a new use.

This article argues that evaluation of dynamic reconfiguration is a *quantitative* rather than simply *qualitative* issue as described as follows: (1) keeping a system operational under reconfiguration (Dowling & Cahill, 2001a,b; Kramer & Magee, 1990); (2) supporting changes to an application's structure and functionality at runtime without shutting it down (Fung & Low, 2009); and (3) integrating new components, or modifying or removing existing ones while a system is running (Soria et al., 2009). The non-stop feature (availability) is just a qualitative aspect and far from an appropriate ground for successful dynamic reconfiguration. The premise that dynamic reconfiguration is more suitable for software evolution lies in its promotion for the QoS assurance of a RSUR. Otherwise static reconfiguration is more acceptable in terms of simplicity.

The limitations of the state-of-the-art as compared with a systematic evaluation are:

1. there is no set of unified criteria to evaluate the impact of dynamic reconfiguration on the QoS of a RSUR;
2. there is no evaluation benchmark covering a rich set of available QoS scenarios;
3. there is no abstraction or representation for modeling available reconfiguration approaches in a unified evaluation context;
4. there is no evaluation on the QoS of a RSUR with respect to the common QoS metrics: throughput and response time;
5. there is no identification or comparison of the QoS assurance capabilities offered by alternative approaches.

Reconfiguration approaches have been evaluated with respect to isolated criteria (as reviewed in the next section), but they have not been realized into a unified evaluation context. Therefore, little quantitative comparison in terms of impact on the QoS of RSURs between these approaches can be found in the current literature.

This article addresses these gaps by proposing an evaluation method and an evaluation benchmark, and conducting a set of experimental evaluations. The stimulus for this article is prior work (Li, 2009) which explores QoS assurance for dynamic reconfiguration of dataflow systems. The novelty of this article with respect to the prior work lies in that it develops the original evaluation ideas into a more systematic evaluation. Thus the novel contributions of this article are to:

1. propose fundamental QoS characteristics as the criteria to evaluate QoS assurance capabilities of available dynamic reconfiguration approaches;
2. propose criteria for discovering the QoS assurance capability of a given approach;
3. propose a representation model to model abstractly available dynamic approaches so that they can be evaluated and compared on a unified evaluation context;
4. introduce a more complex (therefore more realistic) reconfiguration scenario to accommodate a richer set of reconfiguration problems;
5. design an evaluation benchmark to expose existing QoS problems and facilitate the quantitative measurement of QoS in terms of throughput and responsiveness;
6. evaluate and compare the QoS assurance capabilities of existing dynamic approaches.

The remainder of this article is structured as follows. A detailed review of related work in particular with respect to evaluation of dynamic reconfiguration approaches is presented in Section 2. A set of evaluation criteria for QoS assurance capabilities is proposed in Section 3. A qualitative classification of available reconfiguration approaches is presented in Section 4. A set of criteria for discovering QoS assurance capabilities of existing reconfiguration approaches is proposed in Section 5. An evaluation benchmark and its two architectural prototypes are presented in Section 6. New reconfiguration strategies are proposed as abstraction and realization of available approaches in Section 7. The reconfiguration strategies are evaluated and compared for their QoS assurance capabilities in Section 8. Section 9 concludes by outlining the QoS evaluation results arising out of this research.

2. Related work

Related work varies with respect to the kind that dynamic reconfiguration approaches are evaluated.

2.1. Individual evaluation

Some dynamic approaches have been evaluated only with respect to local, isolated criteria. In Bidan et al. (1998) a simple client/server application was used as a case study to evaluate the proposed reconfiguration algorithm for its overhead as measured by reconfiguration time. The case study has limitations making it unsuitable for more inclusion in a more extensive evaluation, due to a lack of complexity, flexibility, and coverage of existing QoS problems. Ajmani et al. (2006) evaluated the reconfiguration framework *Upstart*. To evaluate the overhead imposed by a so-called upgrade layer dedicated to dynamic reconfiguration they tested 100,000 null RPC (Remote Procedure Call) loopback calls and crossover calls; 100 TCP transfers of 100MB data were conducted. The time consumed by the operations was used to compare the performance between the base framework (without upgrade layer) and Upstart. The results showed that the upgrade layer introduced only modest overhead. Truyen et al. (2008) reported the reconfiguration time for adding, removing and replacing a fragmentation service for their case study, an Instant Messaging Service. The overhead of introducing a reconfiguration proxy was reported to be insignificant, but it is not clear to what extent this result generalizes.

The most recent evaluations of reconfiguration overhead include Leger et al. (2010) and Surajbali et al. (2010). Both reported overhead at reconfiguration time. The former compared reconfiguration with and without Java RMI (Remote Method Invocation) transactions, and the latter compared reconfiguration with and without COF (Consistency Framework).

Two important works in dynamic reconfiguration are *quiescence* (Kramer & Magee, 1990) and *tranquility* (Vandewoude et al., 2007). Quiescence refers to a system state where a node (software component) is not involved in any transactions and will neither receive nor initiate new transactions. Quiescence is proved by Kramer and Magee (1990) to be a sufficient condition for preservation of application consistency during dynamic updates. Improving on quiescence, tranquility exploits the notion of a minimal passivated set of components for reducing disruption on the ongoing transactions. Although Vandewoude et al. theoretically analyzed that tranquility has less disruption than quiescence, they did not conduct any quantitative comparisons between tranquility and quiescence in terms of disruption to the QoS of a RSUR. This is evidenced by the following. In Vandewoude et al. (2007), a tranquility-based