The Journal of Systems and Software 110 (2015) 205–221

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

# Using simulation to evaluate error detection strategies: A case study of cloud-based deployment processes

Jie Chen[a,e,*], Xiwei Xu[b], Leon J. Osterweil[c], Liming Zhu[b,d], Yuriy Brun[c], Len Bass[b,d], Junchao Xiao[a,f], Mingshu Li[a,f], Qing Wang[a,f]

[a] Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing, China
[b] NICTA (National ICT Australia), Australian Technology Park, Eveleigh, Australia
[c] College of Information and Computer Sciences, University of Massachusetts, Amherst, MA, USA
[d] School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
[e] University of Chinese Academy of Sciences, Beijing, China
[f] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

*Article history:*
Received 2 December 2014
Revised 12 July 2015
Accepted 25 August 2015
Available online 6 September 2015

*Keywords:*
Process modeling
Simulation
Deployment process

ABSTRACT

The processes for deploying systems in cloud environments can be the basis for studying strategies for detecting and correcting errors committed during complex process execution. These cloud-based processes encompass diverse activities, and entail complex interactions between cloud infrastructure, application software, tools, and humans. Many of these processes, such as those for making release decisions during continuous deployment and troubleshooting in system upgrades, are highly error-prone. Unlike the typically well-tested deployed software systems, these deployment processes are usually neither well understood nor well tested. Errors that occur during such processes may require time-consuming troubleshooting, undoing and redoing steps, and problem fixing. Consequently, these processes should ideally be guided by strategies for detecting errors that consider trade-offs between efficiency and reliability. This paper presents a framework for systematically exploring such trade-offs. To evaluate the framework and illustrate our approach, we use two representative cloud deployment processes: a continuous deployment process and a rolling upgrade process. We augment an existing process modeling language to represent these processes and model errors that may occur during process execution. We use a process-aware discrete-event simulator to evaluate strategies and empirically validate simulation results by comparing them to experiences in a production environment. Our evaluation demonstrates that our approach supports the study of how error-handling strategies affect how much time is taken for task-completion and error-fixing.



## 1. Introduction

Understanding and evaluating complex real-world processes are made more difficult by the challenges in understanding how strategies for diagnosing and repairing errors affect the results produced by these processes. For example, domain experts intuitively know that human-executed process steps are relatively slower and more error-prone than automated steps, but might be more amenable to interactive error diagnosis and the prevention of overreaction by automated error recovery systems. Intuition also suggest that scripts can reduce errors and perform steps much faster than humans can, but that the use of scripts can propagate errors much more quickly and make error diagnosis more difficult. Similarly, common sense suggests that meticulously verifying the outcome of each step at a lower-level of granularity in an operational process helps to prevent downstream failures, but that doing so is expensive and can slow down the overall process. Informal guidelines are widely used in some application domains to decide which steps should be performed by humans and which by scripts, but such guidelines are often weakly justified. Such weakly justified guidelines are also often used to decide how frequently step outcomes should be verified and validated. Experts in other process domains similarly use other weakly justified guidelines.

These guidelines, and common knowledge should be justified, or replaced, by careful studies that provide clear, carefully reasoned justifications for specific approaches and practices. It is particularly important to provide justifications for practices relating to the

* Corresponding author at: Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing, China. Tel.: +86 13466401225.
*E-mail addresses:* chenjie@itechs.iscas.ac.cn (J. Chen), Xiwei.Xu@nicta.com.au (X. Xu), ljo@cs.umass.edu (L.J. Osterweil), Liming.Zhu@nicta.com.au (L. Zhu), brun@cs.umass.edu (Y. Brun), Len.Bass@nicta.com.au (L. Bass), xiaojunchao@itechs.iscas.ac.cn (J. Xiao), mingshu@itechs.iscas.ac.cn (M. Li), wq@itechs.iscas.ac.cn (Q. Wang).

http://dx.doi.org/10.1016/j.jss.2015.08.043
0164-1212/

detection and correction of errors in complex processes. This paper presents one approach for providing justifications to such guidelines and knowledge. This approach uses error-seeding and discrete-event simulation to evaluate the effectiveness of error-detection and correction strategies. We illustrate and evaluate our approach using real-world error-prone processes employed in the domain of cloud computing.

Cloud computing processes (such as those related to the deployment, upgrade, failover, and reconfiguration of cloud-based applications) are particularly appropriate as evaluation vehicles because they are complex and error-prone. These processes are often orchestrations of intricate interactions among cloud infrastructure entities, application software, tools, and human activities. This complexity and the reliance on humans to make key decisions in time-critical situations make these processes particularly error-prone. Moreover, pressures for evolution of the underlying applications, and the emergence of continuous deployment practices are resulting in the need to exercise these processes as frequently as tens of times a day. The challenges of orchestrating these interactions at such high frequency and under uncertainties that are inherent to cloud environments can be considerable, increasing still further the propensity of such processes to error (Zhu et al., 2015). Errors can necessitate additional operations such as time-consuming troubleshooting, undoing steps, and problem fixing and redoing the undone steps. These operations can be expensive and error-prone themselves. To deal with this propensity for errors, these processes typically incorporate strategies for detecting, diagnosing, recovering from, and preventing errors. But the performance characteristics of these strategies can be subtle and hard to fully understand. For example, automated error detection and tolerance may reduce error rates and detect errors earlier but can also mask accumulative subtle errors leading to major outages and making error diagnosis more difficult. And fully automated *overreaction* to an initial small error is often the cause of major failures (Yuan et al., 2014). Humans, on the other hand, may spot errors that computers may miss, but are often slow, which can impede system progress. Therefore strategies for synthesizing these two approaches should be considered carefully as they yield trade-offs between efficiency and reliability. Choosing the wrong strategy may result in wasted resources or errors that propagate unnoticed. There is a need for research on approaches for effectively deciding on appropriate and effective strategies.

These challenges become particularly acute when dealing with large-scale applications that run in distributed cloud environments. The execution platform for a modern large-scale cloud-based system might consist of thousands of nodes, and the maintenance of such a system may require dozens of changes (e.g., the incorporation of new versions of software utilities) a day (Etsy, 2013), each of which may require the execution of a complex process of collaboration between automated tools and a busy operations team. As a consequence, errors are frequent. According to Gartner, "Through 2015, 80% of outages impacting mission-critical services will be caused by people and process issues" (Colville et al., 2010). Some errors arise from faulty system executions, which then trigger operator reactions, such as executing a complex remediation process that itself might be flawed. Errors of this kind, and their cascading effects, may have a significant impact on overall operational costs.

The deployment of cloud-based applications relies on the smooth performance of such key processes as preparing the environment, loading pre-baked virtual machine images into the environment, applying and propagating the necessary configurations, activating and deactivating the new and old versions, conducting small-scale canary testing, and rolling execution images out to perhaps thousands of nodes. All of these processes are complex, error-prone collaborations between human operators and automated systems. They are difficult to test using traditional testing approaches, despite attempts to treat them like regular applications by the Infrastructure-as-Code movement. It is important that errors be identified and handled within minutes or seconds, especially in the case of high-speed, high-capacity systems in domains such as finance, healthcare, and transportation, all critical components of key societal infrastructure. Although humans and automated tools collaborate in performing these processes and in dealing with errors, strategies for this collaboration are presently only guessed at.

We address these challenges by creating a general framework for evaluating error-detection and correction strategies, and then applying the framework to the domain of cloud computing. Specifically we

- *created a framework to integrate approaches for error detection and repair into complex processes. These approaches have characteristics that are demonstrably superior to current best practices, which are often simply weakly justified guidelines based on anecdotal observation and experience.*

And then, we

- *used this framework for sample complex cloud-computing processes, which, unlike the (presumably) well-tested software systems whose deployment they manage, are neither well understood nor well tested.*

In this paper, we model some example deployment operations as processes, each consisting of a collection of steps. Each step is executed by an agent, who is either an automated script, an assistive tool, or a human. Each step requires different amounts of time and various resources, such as computing power, a readied environment, or cloud computing nodes. We focus on two complex and representative processes, deployment and rolling upgrade, to illustrate our approach. We augment an existing process modeling language to define these error-prone processes, and use it to model precisely when errors can occur, the types and distributions of those errors, and the processes involved in checking for and correcting these errors. Recognizing that the error-checking and correcting processes themselves can both miss some errors and themselves actually create other errors, our framework supports modeling these situations as well.

We use a process-aware discrete-event simulator to show the overall effects that strategies can have on the final outcomes of process execution, and to suggest improvements to the processes. We carry out our simulations within a framework that incorporates detailed and precise models of these processes, populated with empirical data and measures obtained by observing real-world process executions. Our simulations are designed to represent realistic error-detection and repair scenarios that take place in the real world, and to support accurate comparisons of different approaches for dealing with these scenarios. For example, we use our framework to answer questions such as "How frequently should an error-checking action be performed, and at what level of granularity?" and "How much does increasing error-checking frequency reduce the risk of system failure?" Our view is that answers to such questions lead to cost-benefit trade-off analysis that could help developers and operators select policies that positively affect system operational costs and product quality.

Finally, we validate our framework and approach by comparing the models and results obtained from simulation studies to observed measurements of multiple large-scale executions of the processes in real cloud computing settings on the Amazon Web Service (AWS)[1] platform. Our results demonstrate that the simulations make reasonable predictions that can help actual operations personnel to conduct what-if analyses to support making better decisions. This, in turn, supports our view of the effectiveness of our overall framework and approach.

The rest of the paper is structured as follows. Section 2 introduces our modeling approach and Section 3 describes our simulation

[1] http://aws.amazon.com/.