Understanding and Debugging System Configuration

Henry Lieberman
MIT Media Laboratory
20 Ames St
Cambridge MA 02139
lieber@media.mit.edu

Earl J. Wagner
Northwestern Computer Science
1890 Maple St, 3rd Floor
Evanston IL 60201
ewagner@cs.northwestern.edu

INTRODUCTION

It is a daily challenge for system operators to keep track of the changes made to a system. In fact, recent investigation has found that system operators often have incorrect mental models of the systems they work with[2]. Managing a system well becomes increasingly difficult when other people or worse, automated scripts or agents, are also changing a system's configuration. Yet much of the potential of Autonomic Computing lies in automating many of the steps of reconfiguration that operators perform, both to increase performance and responsiveness to change, and also to ensure that those changes are performed predictably[1]. Even farther out is the possibility of systems reconfiguring themselves in response to changes in their environments.

Operators now use scripts to manage the configurations of systems but, without extreme discipline, changes to one component may cause unexpected interactions with other components. As with software in general, local changes can have global effects. Now, when an operator observes something unusual in a system log, he must get help from documentation, his notes and memory, or other people including other operators, to identify which configuration change may have produced it. Little attention, however, has focused on presenting, at a high-level, the history of changes made to a system, by whom, and with what goals, in order to explicitly support the identification of causes of errors.

Recent work has found that improved tools for visualizing a system's configuration would decrease the time required by operators to diagnose and repair failures[3]. It articulated the need for fundamental advances in tools to help operators understand system configuration and component dependencies. Those working in software development can see problems similar to the ones that they often face.

The problem of operator understanding is exacerbated by the very agency proposed within Autonomic Computing. User interfaces for autonomic systems might seem like an oxymoron – If the computer is so good at self-regulation, why would the user need to interact with it at all? One answer is that user intervention will take place primarily when things go wrong. No matter how autonomic a system is, unanticipated situations will occur that require advice and direction from a human. The computer cannot automatically divine the operator's expectations and intentions for the behavior of the system, and often its "intelligent" behavior or recovery may create even more problems. The user interface issues for tools then become:

Explanation How can the system explain to the user what is happening in terms that the user can understand and appreciate? How can the system provide visualizations of system operation that give the user an overall feel for what is happening?

Localization How can the user understand which parts of the system are responsible for desirable or undesirable behavior and might need to be changed?

Instrumentation How can the user monitor behavior and history of particular parts of the system?

Repair How can the user determine what needs to be changed and the consequences of proposed changes?

Our approach is to treat these problems by analogy with software debugging. We provide tools for explanation, visualization, localization, instrumentation and repair of high-level processes instead of low-level program code. More generally, the situation of an operator trying to understand the system's configuration is parallel to that of a software developer trying to understand a large software project. Both individuals must understand the current system state in terms of its most recent execution, as well as the history of its modifications.

We will present an agent we've developed called Woodstein that supports users in understanding the relationship between a system's current state and its history. Though just a prototype, Woodstein records the actions taken by the user and other entities. By matching these records to action models, it tracks the current status of an action and whether it has been successfully completed. A user can debug a failure by working back from the output to the original inputs that may have played a role.

REFERENCES

1. IBM Corporation. Autonomic computing manifesto, 2002.

2. Dennis G. Hrebec and Michael Stiber. A survey of system administrator mental models and situation awareness. In *ACM SIGCPR, San Diego,* April 2001.

3. David Oppenheimer, Archana Ganapathi, and David A. Patterson. Why do internet services fail, and what can be done about it? In *4th USENIX Symposium on Internet Technologies and Systems (USITS '03),* 2003.