An Analysis Method for Software Deployment Conflict Detection

Han Wu, Xinhui Li, Jiawei Hu, Jingsen Zhang, Yucheng Hua, and Lei Qiu
North China University of Technology, Information Institute, China

Abstract. This paper proposes an analysis method for software deployment conflict detection. First, we analyze the possible dependencies in the process of software deployment. Based on these dependencies, we find out the factors that may lead to software deployment conflict.

1 Introduction

Many scholars and research institutions are committed to deployment conflict detection and configuration error detection, diagnosis and fault repair related technologies and methods to improve the availability and reliability of complex application systems.

Li BingPeng Et al. [1] proposed a model of software intelligent deployment and deployment conflict detection. This software intelligent deployment model mainly studies software intelligent deployment which can automatically adjust the software deployment process according to different environments.

Zhang Jian Et al. [2] proposed an automatic deployment and monitoring system for large-scale deployment software. It adopts a relatively mature three-tier architecture, and improves the system reliability, stability, scalability, etc. by specifying the interface, interactive data format and other methods.

Sotiropoulos T [3] Et al. Proposed a method to detect missing dependencies and notifiers in a puppet program. This method collects the system calls of the puppet program calls, and models them in FStrace. Through FStrace, it captures how the advanced programming structure interacts with the operating system and the relationship between them.

Zhangsai [4] Et al. Put forward an automatic diagnosis technology of software configuration error. This method is based on the comparison of the behavior of related predicates between execution, so as to achieve the effect of fully automatic, diagnosis of crash and non-crash errors and no need of operating system level support.

There are two reasons why the existing software deployment system cannot automatically analyze the output conflict:

1. In the software deployment, only the deployment target is obtained, but the constraints and dependencies between the deployed software are not considered in the software deployment, resulting in software deployment conflicts.

2. In the process of software deployment, only the dependency relationship between deployed software is described, but before the software deployment, constraints and dependency problems are not detected in combination with the situation of deployment target. Constraints and dependency problems are only detected after the software is deployed to the deployment target, resulting in software deployment conflicts.
This algorithm enables users to detect possible software deployment conflicts in advance before installing the software, and reconfigure the local environment according to the error reasons given by the algorithm, so as to avoid the loss caused by dependency errors during the deployment process.

2 Approach

2.1 Dependencies in software deployment

In software deployment, the dependence from the perspective of algorithm can be divided into three parts: platform dependence, hardware dependence and software dependence.

This algorithm uses recursion idea to obtain software dependency and check one by one until there is no less dependency, so it is divided into three parts: input, recursion analysis and output, recursion analysis is divided into two parts, to obtain software dependency and check, and also includes dependency error output.

2.2 Conflict detection in software deployment

The implementation of the algorithm is divided into three parts: input, recursive analysis and output. The flow includes three steps.

(1) Input
There are three types of information to enter:
① The name of the target software to install.
② Version number of the target software.
③ If you need to change the default storage location of the local installation package, you also need to enter the destination folder address.

After obtaining the necessary information, we need to build software nodes, which should include storage location, name, version and other attributes.

(2) Recursive analysis
Recursive analysis is divided into the following two steps:

① Get current software dependencies:
The tasks to be completed in this step are as follows:
i. Extract the found puppet resource package.
II. Find the metajson.data documents.
Ⅲ. correctly read the software dependency information, extract the name and version range of the dependent software, and then build a new software node, step ②.

IV. according to the installable flag in the dependency software node after step ②, output whether the software can be installed correctly. If it can be installed correctly, add it to the installation sequence list; otherwise, it will be prompted that the dependency of the software cannot be obtained correctly from the local.

② Check if the local contains qualified dependent resources:

i. Obtain the storage location and software name from the dependent software node, and find the puppet resource package containing the software name in the name according to the two information; if the package containing the software name cannot be found, report that the local environment does not have the corresponding software prompt, that is, XX dependency is missing, and return to the previous step.

II. Judge the version according to the file name of the puppet resource package (the general format of the file name of the puppet resource package is Puppetlabs-software name-software version. tar.gz), set the installable flag in the node information of the
dependent software to true if the dependency is proved to be satisfied within the required version range, Call ① to obtain the dependency of the current software, and then return it; otherwise, if it is proved that the local dependency exists, but the version does not match, you should prompt "XX dependency version does not match", and return to the previous step.

For recursion, the dependent installable software will be added to the installation sequence list from bottom to top. If there are conflicting dependencies, or repeated dependencies, they will be adjusted in the output step.

(3) Output
It is necessary to adjust the previously obtained installation sequence:
For duplicate dependency: If duplicate dependency with the same name appears in the sequence, we keep the first one;
For conflict dependency: We still select the installation that appears first and has the deepest depth in the dependency tree.
To sum up, when building the installation sequence, we only need to use the software name as the judgment standard. When adjusting the installation sequence, we only need to select the first dependency in the sequence and eliminate the subsequent dependency with the same name.

3 Experiment
The experiments in this paper are mainly carried out on the puppet platform and Linux operating system.

3.1 Purpose of the experiment
This case test mainly includes the following two purposes:
(1) Under the environment that can complete the local installation correctly, check whether the software dependency of the software to be deployed is complete and correct, and give the correct installation sequence;
(2) Under the premise of missing dependency, too new or too old dependency version, existence of dependency with the same name and conflict dependency, the corresponding error information can be clearly given.

3.2 Subjects
The deployment software used in this test is Apache 5.4.0 in the puppet library, and its related dependencies are shown in Figure 2:

![Figure 2. Apache 5.4.0 and its dependencies.](image-url)
3.3 Experimental results and analysis

Experiment 1: lack of a certain dependence
Environment: missing dependent concat for apache
Error reported in operation result
Because the storage location of the package does not contain Apache’s dependency concat, the error message first prompts "unable to obtain the correct dependency", and the puppet resource package with the name containing concat cannot be found in the storage location, and finally gives the solution—Download with the instruction of "puppet module install"

Experiment 2: a dependent version is too low / too high
Environment: stdlib version is too low, it should be above 4.13.1
Operation result error
Because the stdlib version in the storage location is 4.9.0, and the stdlib dependency required in Apache 5.4.0 must be $\geq 4.13.1 < 7.0.0$, the stdlib version in the experimental environment is too old, so the error type "incorrect stdlib version" is indicated in the error message first, and then the required stdlib version range is given.

Experiment 3: whether multiple files can select the right qualified software for installation detection

Environment: Apache 3.0.0 / 5.4.0, stdlib 4.9 / 6.3, followed by the version to be installed
Installation sequence given by operation results:
stdlib-6.3.0 translate-2.2.0 concat-6.2.0 apache-5.4.0

In the experimental environment, we prepared Apache 3.0.0 and stdlib 4.9.0 as interference items, both of which do not meet the installation requirements. During the running process of the algorithm, because the version of Apache 3.0.0 is not the same as that of the target software, the algorithm directly finds out Apache 5.4.0. Although stdlib 4.9.0 is also searched, but because the version is not the same, the algorithm still gives the same error report as experiment 2. From the installation sequence given by the last algorithm, stdlib 4.9.0 is not added to the correct installation sequence. Finally, the installation sequence given by the algorithm is consistent with that reflected in the dependency tree of Apache 5.4.0.

4 Summary and prospect

This paper proposes a conflict detection method for software deployment, which can detect whether the local installation environment can support the correct deployment of the target software. If it can, the correct installation sequence is given to eliminate the dependency with the same name and conflicting dependency; if the local environment does not support it, the algorithm can detect the problem of missing dependency and wrong dependency version, and provide the corresponding solutions and correct version range.

This work was supported in part by Talent Special (218051360020XN115/016), Basic Research Business Fee (110052971803/038) Plan, Project of Beijing College Students’ Scientific Deep Research and College Student Research Training Program (218051360020XN214).

Reference